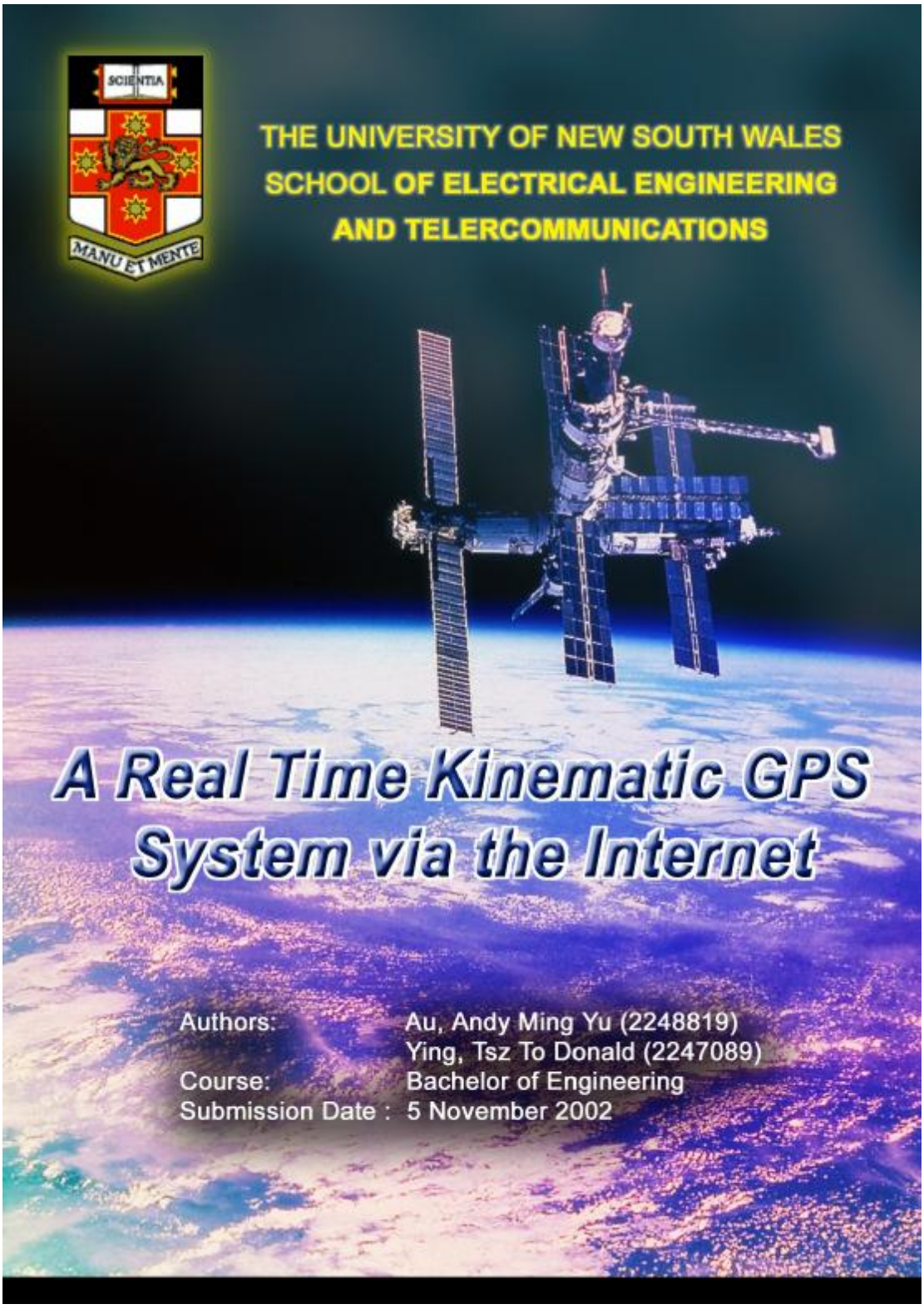

SCIENTIA
MANU ET MENTE
THE UNIVERSITY OF NEW SOUTH WALES
SCHOOL OF ELECTRICAL ENGINEERING
AND TELERCOMMUNICATIONS
A Real Time Kinematic GPS System via the Internet
Authors: Au, Andy Ming Yu (2248819)
Ying, Tsz To Donald (2247089)
Course: Bachelor of Engineering
Submission Date : 5 November 2002

Abstract

Real Time Kinematic (RTK) is a high precision GPS-based positioning technique that requires raw carrier phase data to be transmitted from a master GPS receiver to the rover GPS receiver via a data link with very low latency. This data link could be appears in different forms, such as GSM, GPRS, Internet or simply a Serial cable. Each of them has its own advantages and disadvantages.

Internet is chosen as the data link between the GPS receivers in this thesis project. Hence, investigation would be performed on the protocols and hardware for using the Internet to deliver such data to the rover receiver. This document outlines the background theory, concepts and testing of the Real Time Kinematic System.

Table of Contents

1. Introduction…… 5

2. Introduction to Global Positioning System…… 6

2.1 How does GPS work…… 6

2.2 GPS satellite signals……... 8

2.3 How accurate is GPS…….. 8

3. Introduction to Data Networking…… 11

3.1 Open Systems Interconnection (OSI Reference Model) ..… 11

3.2 Transmission Control Protocol / Internet Protocol (TCP/IP).. 14

4. Introduction to smartGPS TINI board……... 16

4.1 TINI board hardware details / Specifications..……. 16

4.1.1 The TINI Chip set……. 16

4.1.2 The TINI Runtime Environment……. 16

4.1.3 Supported Applications Programs Interfaces (APIs)…. .18

4.1.4 The TINI Java Virtual Machine…….. 18

4.1.5 TINI System Architecture…… 18

4.1.6 TINI Software Development…… 19

4.1.7 Compiling and Running in Java Code on TINI…….. 20

4.2 DGPS and RTK using smartGPS…… 20

4.3 smartGPS User Interface…….. 21

4.4 Flow Diagram of User Diagram…….22

5. Introduction to Java programming…….. … 23

5.1 Object Oriented Language……..… 23

5.2 Tradeoffs…… 23

5.3 Network Savvy…….. 24

5.4 High Performance……..... 25

6. Possible applications…… 26

6.1 Courier Shipment Tracking Services……. 26

6.2 Allocation of Engineering Resources……..... 27

6.3 Recreational Users……..…. 28

7. Research on Real Time Kinematics GPS…….. 29

7.1 Background to DGPS…….……. 29

7.2 DGPS Positioning over the Internet……..……. 30

7.3 The Internet Protocol best suits DGPS……..…… 31

8. Real Time Kinematics……..…….. 32

8.1 RTK Principle……..……..……. 32

8.2 Temporary Master Station……... ……..……. 33

8.3 Permanent Master Station…….……..…… 33

8.4 Network RTK……..…… 33

8.5 Different approach on DGPS and RTK…..…… 34

8.5.1 EUREF……..…… 34

8.5.2 Differential and RTK corrections for the Internet Car.... 35

8.5.3 RTK Positioning using Internet Protocol and GPRS.… 36

8.5.4 SWEPOS……..…… 37

9. Communicate between GPS receivers..... 38

9.1 Use of Radio Modem……..……. 38

9.2 Converting UDP Packet……..……. 39

9.3 UDP Packet Travel Time……..…….. 41

9.4 Real Time Kinematics GPS System Requirement.....…… 42

9.5 Analysis TCP/IP and UDP Traffic Software……..…… 44

9.6 Packet Travel Time Measurement……..……... 46

9.7 Packet Travel Time Analysis……..……. 48

10. GPS Applet Controller…… 52

10.1 GPSolution……..…… 52

10.2 TINI LCD Screen……..…… 53

10.3 GPS Applet Controller…… 54
10.4 Communicate with the Receiver…….. 55
10.4.1 Fix Master Receiver Location……... 56
10.4.2 Logging Master Receiver Data……. 56
10.4.3 Set DGPS Timeout……. 57
10.4.4 Receive Data from Master Receiver…… 57
10.4.5 Logging Rover Receiver Data…….. 58
10.4.6 Display Rover Location…….. 58
10.4.7 Manually Enter Command……. 59
10.5 Editing the GPS Applet Controller……... 60
10.6 Accuracy of the Computed Location……... 60

11. Thesis Promotion…… 62
11.1 Website…… 62
11.2 Promotional CD…….. 63

12. Further Development…… 64
12.1 Additional Serial port and Address space…….. 64
12.2 Host Internet Server…….. 65
12.3 Display Features…… 65
12.4 Further Experimentation……... 65
12.5 Remove Unnecessary Data before Transmitting……. 66
12.6 AppleTalk…… 67

13. Conclusion…… 68

14. Reference……... 69

15. Appendix A Standard Command Tables……. 73
B Developed Software…….. .. 78
C Logs Summary……..…….. 89
D Selected Experimental Results…….. 91
E Operate smartGPS TINI board…….. 93

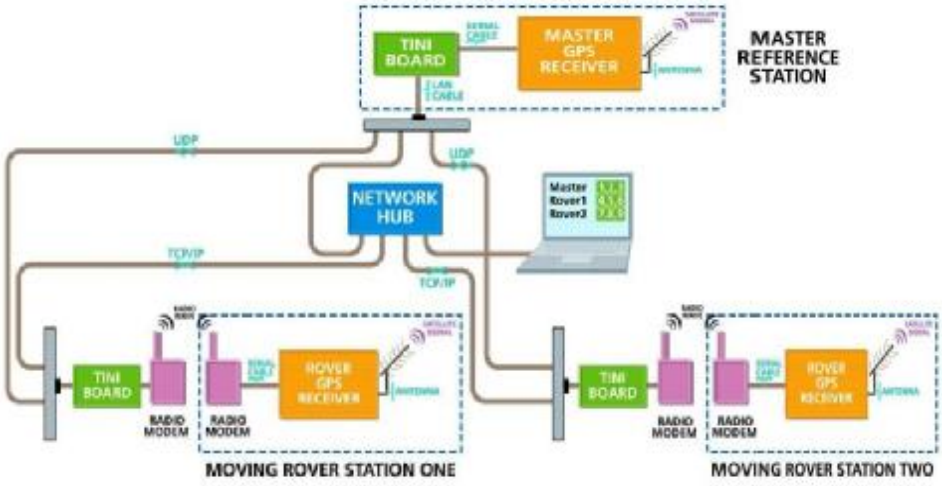
1. Introduction

The aim of this thesis project is to develop a Real Time Kinematic GPS system based on the Internet connection, so accurate rover GPS receiver positions can be found. To achieve that, the data has to be transferred in “real time” to avoid the GPS receiver “time out” problem.

Managing the traffic delay over the Internet would be the biggest challenge of this project. In other words, the key of this project is to minimise the packet travel time from the GPS master receiver to rover receivers. Intensive investigation is performed on Internet based protocols (TCP/IP and UDP), in order to find the best transmission path which satisfies the Real Time Kinematic requirements.

A homepage with Java applet control panel is created to enable users issuing commands to master or rover receivers as well as monitoring the location of rover receivers in real time.

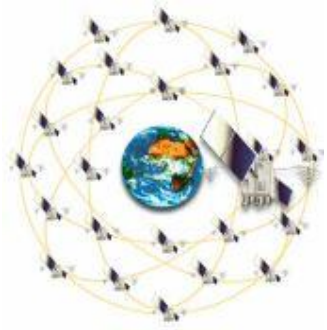
A Real Time Kinematic GPS system via the Internet



2. Introduction to Global Positioning System

Global Positioning System (GPS) is a satellite-based scheme which can provide users with highly accurate latitude, longitude, altitude and velocity measurements. It is available globally at anytime, anywhere and any weather as long as the receiver is in view of the satellites. GPS satellites travel at an amazing speed at about four kilometers per second, even faster than a speeding bullet.

Figure 2.1 - Twenty-four GPS satellites orbiting around the earth



This technology was initially developed by the US Defence Department, one part of this system is reserved for military purposes while the other part can be used by anybody for free simply by the purchase of a GPS receiver.

2.1 How does GPS work

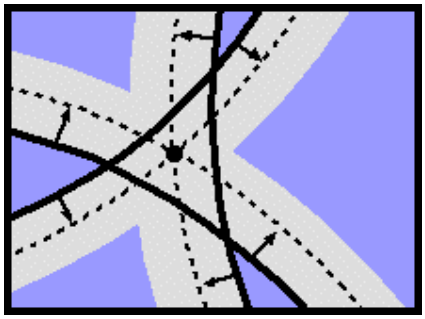
There are 24 GPS satellites orbiting at about 20200 kilometers above the earth's surface, this allows 4 or more satellites in view from anywhere at any instant. These satellites transmit two microwave with frequencies L1 (1575.4MHz) and L2 (1227.6MHz) traveling at the speed of light (approximately 300 000 km/s).

By multiplying the signal's travel time from the satellite to the GPS receiver with the speed of light, we will know the distance between the satellite and the receiver.

[Distance] = [Velocity] x [Time]
[Distance between satellite & receiver] = [Speed of light] x [Signal travel time]

But in order to calculate the distance between them, how can we be sure they both have the synchronized clock? We can't! That is why beside the latitude, longitude and altitude, we have one more unknown – the GPS receivers clock error, totally four unknowns.

Figure 1.2 - The relative size of each sphere



Obviously, four equations can solve four unknowns. Similarly, a GPS receiver can use four or more satellites to calculate these unknowns. By precisely calculate these distances relative to each other; only one possible point can intersect as shown in the diagram above and therefore able to find the exact GPS receiver position.

2.2 GPS satellite signals

We know the satellites are transmitting signals using L1 and L2 microwave frequencies, but what sorts of signals are being sent? Three different signals are sent. The first two are binary codes are for military users (the P-code) and civilians (the C/A-code). The third signal is a navigation message which is available to all users. The purpose of the navigation message is to let the GPS receiver know the satellite is in good condition, so the signals can be used in confidence.

The GPS receiver then used a specialized computer to calculate its location based on these satellite signals. The receiver does not have to transmit anything to the satellite and the satellite does not know the receiver is there. Hence, there is no limit to the number of GPS receivers that can be using the system at any one time.

2.3 How accurate is GPS

In early 1990, the US Department of Defence introduces a policy of degrading the civilian use of the GPS technology called “Selective Availability”. Basically, they decided to scramble the satellite signals so that civilians could not get accurate GPS results. As GPS becomes widely used in safety, scientific and commercial aspects, pressure from world wide made the White House terminated this policy from May 2000.

Nowadays, most hand-held GPS receivers have about 10 meters of true horizontal and about 20 meters true vertical accuracy. Other types of receivers use a method called Differential GPS (DGPS) to obtain much higher accuracy of up to 1 meter which will be explained in later chapters.

2.4 Errors in GPS

The following sources of error could affect the accuracy of the GPS positioning results so that system reliability will be reduced.

[Atmosphere]

When the satellite GPS signal got refracted when it travels through ionosphere and troposphere of the earth. Therefore errors will occur in expected signal travel time and distance between satellite and receiver. Temperature, humidity and pressure are key factors affecting atmospheric error. Fortunately, this kind of error can be minimized by using differential GPS which will be discussed in details in chapter 7.

[Satellite orbits]

Inhomogenity of earth's gravity field, tidal effects from moon and sun, solar pressure and relativistic effects can cause malfunctioning of satellites. So there are orbit biases which will affect the accuracy of GPS. .

[Multipath]

This kind of error caused by the fact that the signal may be reflected by a nearby surface before reaching the GPS antenna. Interference between reflected signal and direct signal will result incorrect carrier phase measured value. Multipath error is hard to avoid and detect indeed.

[Noise]

Random noise always exists in the measurement even all above mentioned errors have been correctly modeled, therefore same position measure can be obtained everytime. Measurement noise is about 0 to 10 meters while pseudorange noise for carrier measurement is 0.2 to 5 millimeters.

[Obstruction]

Building, trees or any other tall and big object can cause obstruction when using GPS outdoor. Poor or even no positioning readings may occur due to obstruction error.

[Clock]

Satellite clock and the receiver clock are very important when calculating the distance between satellite and the receiver. Regardless of the extremely stable performance of the satellite clock, because normal receivers use less expensive quartz clocks, therefore clock errors in the receiver are hard to predict.

3. Introduction to Data Networking

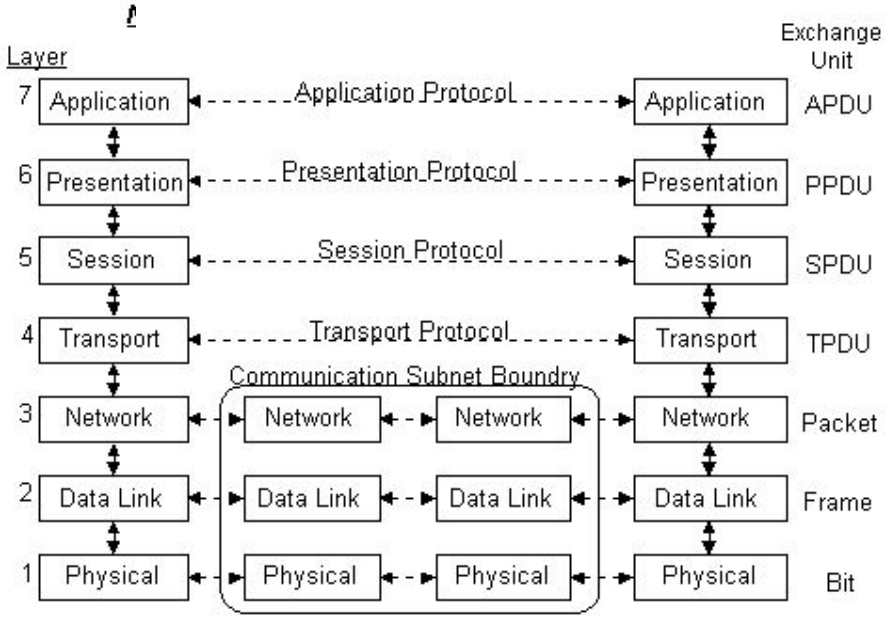
3.1 Open Systems Interconnection (OSI) Reference Model

[Introduction]

The OSI model has seven layers. Each layer performs specific functions and communicates with the layers directly above and below it. Higher layers deal more with user services, applications, and activities, and the lower layers deal more with the actual transmission of information. The purpose of layering the protocol is to reduce the design complexity and separate specific functions.

The OSI reference model is shown below:

Figure 3.1 - The OSI reference model with network architecture



3.1.1 Layers

[Physical layer]

Physical layer Is concerned with the exchange of the individual bits over a medium. Issues such as the how hardware being used, what voltage and timing should represent a 1 or 0 are considered at this layer.

[Data link layer]

Data link layer is responsible to ensure the reliable transmission of information units, packets or frames so that it appears free of undetected transmission errors to the network layer.

[Network layer]

Network layer concerns with controlling the operation of the subnet. Processes like packet routing, congestion control, protocol conversion are involved in this layer.

[Transport layer]

Transport layer provides reliable, transparent transfer of data between terminals. It isolates the upper layers from the inevitable changes in the underlying technologies. Multiplexing, flow control and broadcasting messages can be done at this layer.

The protocol used may be:

[Connection-oriented] – connection established between sender and receiver, messages are sent over a fixed route, the connection is released. Messages arrive, at the destination, in order.

[Connectionless] – information parcels are routed to their destination individually; in order to utilise the most efficient link at the time. Messages may need to be re-ordered at the destination.

3.1.2 Session layer

Session layer provides enhanced services useful in some applications. It manages dialog control and provides the control structure for the communication between applications.

[Presentation layer]

Presentation layer provides certain functions that are requested. It is concerned with the syntax and semantics of the information. A “network standard” presentation is also provided at this layer.

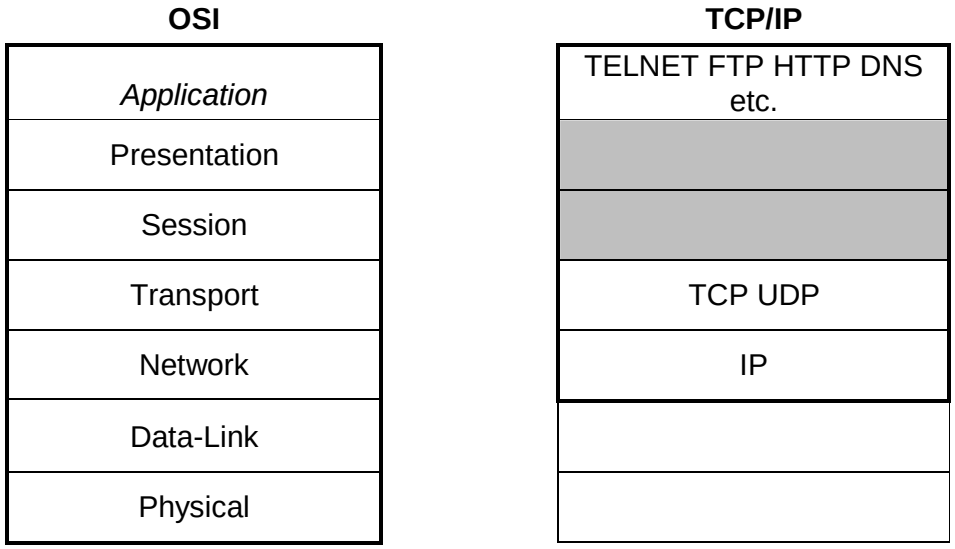
[Application layer]

Application layer works directly with the user or application programs. It provides access to the OSI environment for users. It is called application layer because it contains network applications. For example, electronic mail, file transfers, virtual terminal protocols and distributed system, etc.

3.2 Transmission Control Protocol / Internet Protocol (TCP/IP)

TCP/IP is a suite or protocols which do not have session and presentation layers. it is a standard for communications on the Internet

The difference between OSI model and the TCP/IP is shown below.



[Internet Protocol (IP)]

The Internet Protocol is used at the network layer. Sometimes it referred to as the "Internet Layer" in the TCP/IP model. In IP, each machine or "host" is identified using a 32-bit number called and "IP address".

A header is appended to the data coming from the data-link layer creating an "IP datagram". The source and destination IP address, information about the header and data length, a checksum, and data required to reconstruct the messages are included in the IP header.

Internet protocol performs three main functions, which are routing datagrams through the Internet, identifying data source and destination and restricting the size of datagrams, depending on the network specifications. (J. Kurose and K. Ross, 2001)

[Transmission Control Protocol (TCP)]

TCP is connection oriented protocol, meaning that TCP will set up, maintain, and tear down a connection. TCP keeps track of the status and state of data passing through it. It provides reliable end-to-end data transmission with several mechanisms, like flow control, congestion control, error recovery, and so on. TCP can also multiplex data from different applications and is full duplex.

[User Datagram Protocol (UDP)]

UDP also resides in the Host-to-host layer with TCP. UDP is connectionless and is merely a transport level protocol for the applications in the layer above. Connectionless means that the datagram is sent without first setting up a connection.

UDP does not do any end-to-end error checking but uses a checksum in the UDP header. This checks only the correctness of the header. UDP is used for small data transfer where an error is not a serious problem, for instance, RTK applications. It is faster and cheaper than TCP because it doesn't have the expensive error checking software, nor does it have to take the extra time to error-check.

4. Introduction to smartGPS

SmartGPS is used as an interface between RTK GPS receiver and Internet throughout the entire thesis project. smartGPS was from last year 4th year thesis group. It was build based on the Dallas Semiconductor's Tiny Internet Interface (TINI) Microcomputer. Generally, smartGPS and TINI board mean the same thing.

4.1 TINI board hardware details/specifications

Inside smartGPS hardware, there is a TINI board of which the heart is a Dallas DS80C390 microprocessor, running at a clock speed of 36.8 MHZ.

4.1.1 The TINI Chipset

The TINI board chipset consists of the following hardware:

- Dallas DS80C390 microprocessor
- 10Base-T Ethernet® Network Interface Card
- Real Time clock for time stamping
- 512Kb Flash Read Only Memory for storage and execution of the runtime environment
- 1MByte total SDRAM, with Li^+ Cell an non-volatiser circuit, for RAM and file storage

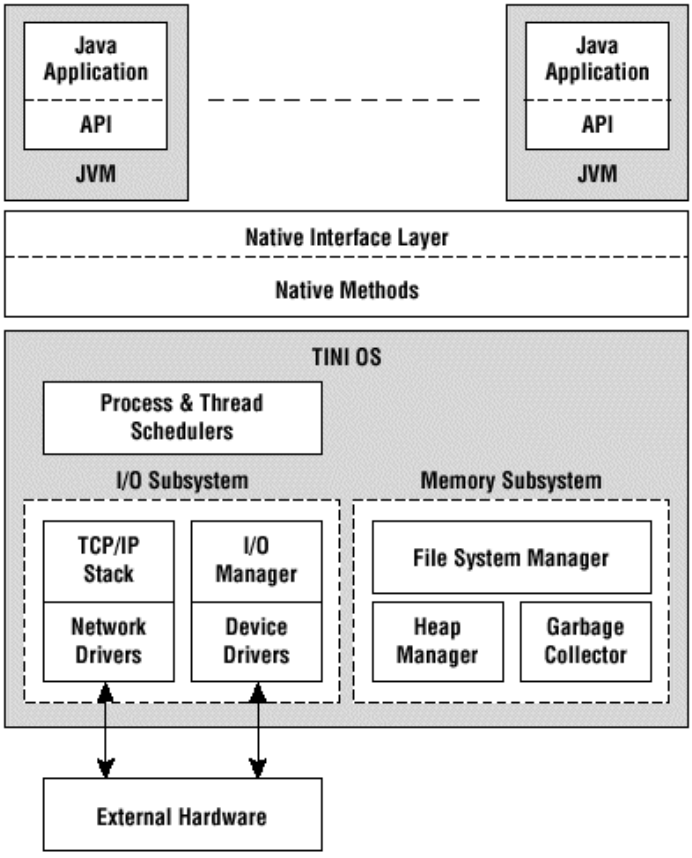
4.1.2 The TINI Runtime Environment

A runtime environment defines the services, provided by software that describes hardware drivers, protocol stacks, APIs and all other functions necessary for developers to create their applications.

The TINI runtime environment is stored in flash ROM so that the system maintains the code in the absence of power. Flash memory ensures that the runtime environment can be easily updated, but can't accidentally be deleted.

The TINI runtime environment is substantial. Care has been taken to ensure that the developers can focus on creating applications, as opposed to providing the layers of infrastructure necessary to execute code. A graphical representation of the TINI runtime environment is illustrated below.

Figure 4.1 - The Layers of the TINI Runtime Environment



4.1.3 Supported Application Program Interfaces (APIs)

The most recent version of Sun's Java Development Kit (JDK) API at the time of writing is version 1.4.1. TINI provides full support for the JDK API version 1.1. Some of the classes from later API versions have been supported, where they have been deemed useful and relevant to TINI applications.

Sun also provides a Java Communications API, which is designed to provide access to communications devices, such as serial and parallel ports. This is supported by TINI.

A third API includes code that is specific to TINI. It provides classes to access hardware and resources, such as the Real Time clock and TINI operating system. It is often considered good practice to minimise the use of TINI specific code. This ensures that programs can be tested on a non-TINI system, and easily ported to other Java embedded microcomputers.

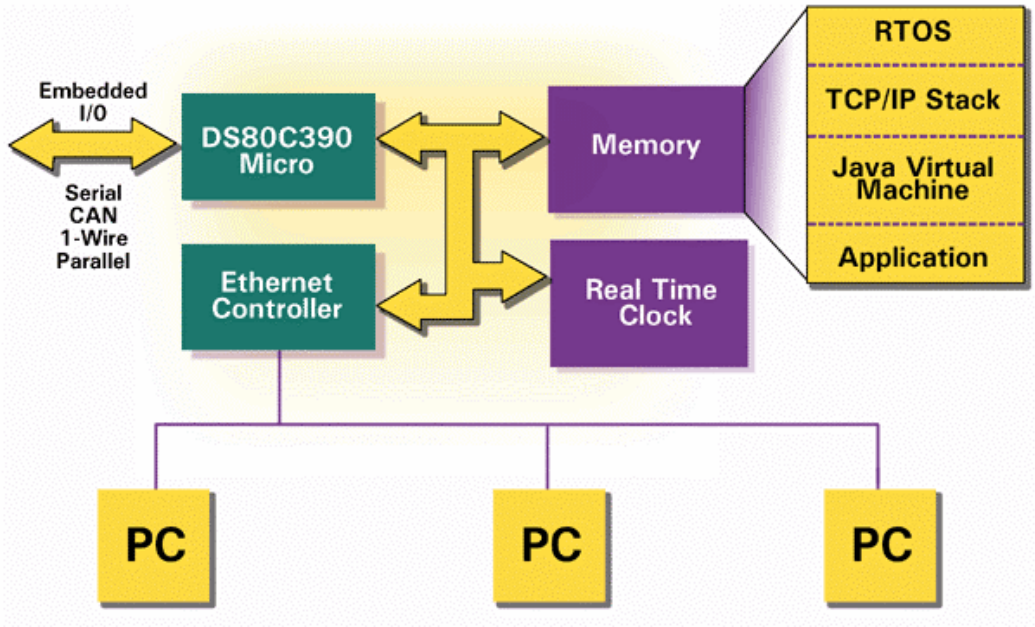
4.1.4 The TINI Java Virtual Machine

The TINI JVM is less than 40Kb in size. All primitive types are supported. So too are threads, although only 16 can be executing at once.

4.1.5 TINI System Architecture

The interaction of the TINI hardware devices, runtime environment and networking interfaces are illustrated in the block diagram on the following page:

Figure 4.2 - TINI System Block Diagram



4.1.6 TINI Software Development

TINI code is developed on a desktop machine, and sent to the TINI when complete. The TINI SDK includes serial, FTP and TELNET servers. By default, commands to start these servers are included in the .startup files. The server facilities make it easy to upload and execute TINI applications

The supported APIs are installed on the TINI at the same time as installing Slush. The APIs also installed on the development machine, in the appropriate directory. This means that code can be compiled, without error, on the development machine, rather than the TINI itself. Hence, TINI hardware costs are minimised and the programmer can develop on a machine with which they are comfortable and familiar.

4.1.7 Compiling and Running Java Code on the TINI

An executable file must be produced from the class files for a program, before it can be run on the TINI. The process for building the TINI executable is similar to zipping up the class files using a compression program. After the executable file is built, it is then sent to the TINI. The build program is called TINIConvertor and is supplied by Dallas Semiconductor. The output file has a .tini extension.

[TiniConvertor can be run as follows]

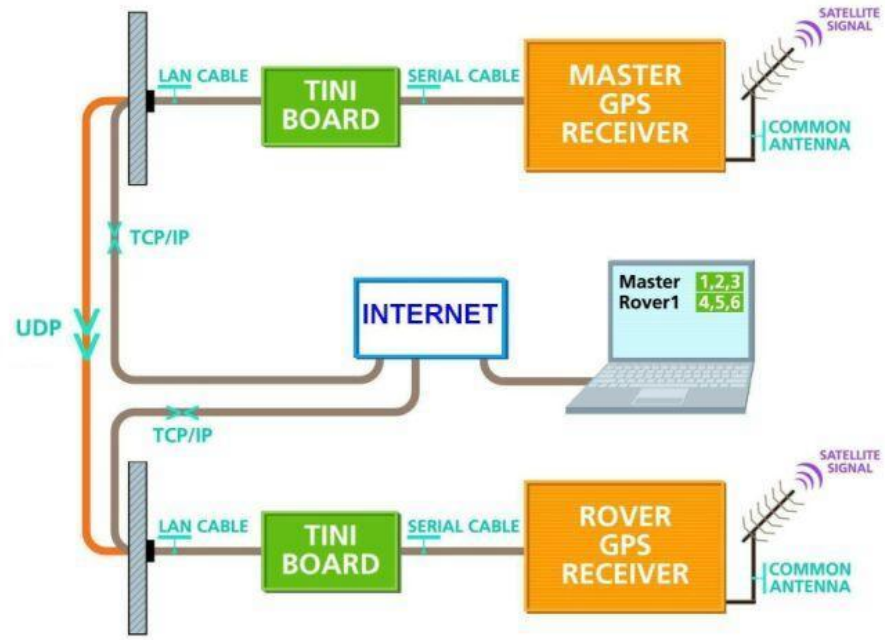
C:\> java TINIConvertor -f Filename1.class -f Filename2.class -f Filename3.class
-o Filename.tini -d <TINI Install Dir>\firmware\tini.db

The usage of the TINIConverter command involves listing each class file after a –f tag, followed by a –o and the name of the output file. Finally, the location of the tini.db file, which describes how files are converted, is referenced with a –d tag.

4.2 DGPS and RTK using smartGPS

SmartGPS provides data communication facilities to the GPS receivers. It has ability to accept messages and transmit them to other machines. Consequently it should have same ability for many GPS units to form a network. However, we would only use the two existing smartGPS units which allow us to do testing between one master GPS receiver and one rover receiver as seen in the figure..

Figure 4.3 - Experimental scenario for a smartGPS Internet RTK DGPS network



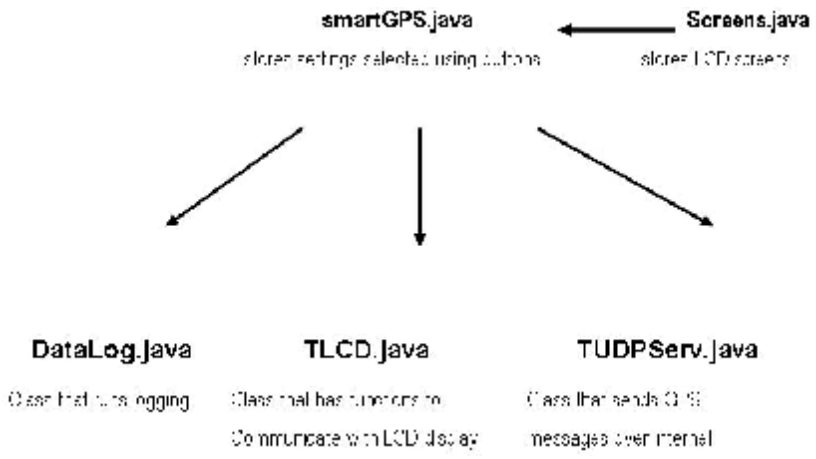
4.3 SmartGPS user interface

SmartGPS is a very user-friendly device that comes along with a LCD, buttons, and a set of simple operations. The LCD display offers a variety of commands that are useful in programming the screen output. It provides intelligence to enable easily adaptable features such as graphical capabilities. The class file TLCD.java is loaded in smartGPS for user to communicate with LCD display. Three input buttons and a fourth reset button have been interfaced on the smartGPS allowing user to select different options for smartGPS operations. They are “Up”, “Down”, “Enter” and “Reset” button.

Java software was required to ensure that the buttons worked in conjunction with the screen to display status information and execute programs. Moreover, the smartGPS is designed to begin run its software when the power supply is switched on.

In order to give feedback to the user, there are several displays that are shown on the LCD screen. These screens are stored as String objects in the Screens.java class. Then they are displayed on the screen using the TLCD.java class. Figure 4.4 has shown the detail interaction of smartGPS class files. These files are crucial in making smartGPS user-friendly.

Figure 4.4 - The main smartGPS program



4.4 Flow Diagram of User Interface

In order to show how user can control smartGPS with the buttons and system menu, several screenshots are shown in appendices. The first, second and the third flow diagrams show available selections under Applications, Diagnostics and Settings options respectively.

5. Introduction to Java Programming

This introduction is meant to clarify the main issues of Java and to separate some of the myths and facts. The purpose of this introduction is not to explain programming constructs, but rather to clarify the purpose of the Java language.

Java was originally planned as a programming language for controller chips and various consumer electronics. Given that a controller on a motor or some device may need upgrading in the future, the idea was to write a language that would be very portable across different hardware. In time, the idea evolved into a language for desktop computers, and in keeping with modern trends, a language which interfaces to the Internet. (Gran, An Introduction to Java)

5.1 Object Oriented Language

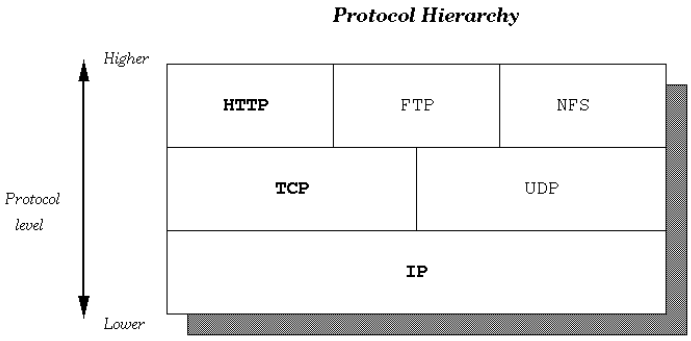
Java is an object-oriented language. The object-oriented model focuses on the data and methods which manipulate the data. Through the use of classes, and other object-oriented constructs, the programmer models the behavior associated with data. The object-oriented approach to programming has the advantage of rapid development and simply code reuse.

5.2 Tradeoffs

The tradeoff for this portability is that Java does not utilize any specific platform strengths, such as OpenGL or QuickDraw 3D. It performs all of it's graphics through the Abstract Windowing Toolkit (AWT). For most simple applications, this is not a problem, but many are yearning for a more robust graphics toolkit, or access to platform-specific routines.

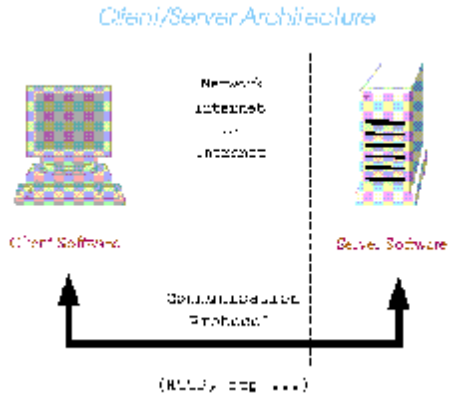
5.3 Network Savvy

Figure 5.1 - The Protocole Stack



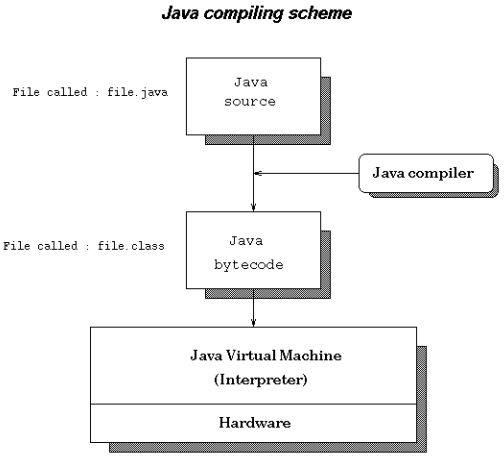
Java comes with a complete library for network applications which make it easier for the programmer to deal with protocol from the lower level like TCP/IP to the higher level HTTP, FTP. A protocol consists of all the rules that two computers or two programs need to know to communicate over a network. (Chancogne & Austin, 1996)

Figure 5.2 - The Client/Server Architecture



Tasks like client/server connections and access to remote objects on the Internet become easy. Opening a specified URL (Uniform Resource Location) somewhere on the net with Java is as easy as opening a local file on the system.

Figure 5.3 - The Java compiling and running scheme



One advantage of this approach is that the Virtual Machine has already been ported to different type of hardware. So any Java program that was written can run on any of those Virtual Machines without any changes. Otherwise someone has to develop a new Virtual Machine for the specific hardware.

5.4 High Performance

Java is a high performance language. Most interpreted languages are roughly twenty times slower than an equivalent compiled program written in C. Java is no exception to this rule, but with the advent of new technologies, such as Just In Time (JIT) compilers and upcoming Java accelerator chips, the gap narrows. For calculation-intense applications, it is best to supplement Java with a compiled language, but this will not be the case in the future.

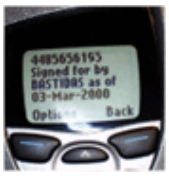
6. Possible Applications

6.1 Courier Shipment Tracking Service

As the world become more globalise, government agencies, companies and individuals increasingly rely on courier services. In the market, there are several courier companies offering similar services. In order to compete with each other, they have to offer fast, responsive and cost-effective deliveries to satisfy customers need.

One of the most important services customers looking for is the shipment tracking service, allowing them to check the exact location of their urgent shipments from collection to delivery at any time.

e-track



sms-track

Steps to track the shipment locations

1) A shipment number and password will be issued when customer post their shipment.
2) Courier company attachs a “RTK GPS system” on the document or parcel and start tracking.
3) Customers can track their shipment using the company homepage (via e-track) or mobile phone (via sms-track acknowledgement) using the shipment number and password issued.

6.2 Allocation of Emergency resources

Everybody knows 000 is an emergency hotline service to contact in life threatening or urgent situations. Every minute, Telstra Communications Centre receives many calls and has to arrange appropriate responses from a local police station or from other services (e.g. Ambulance or Fire Brigade) across Australia.

To response to accidents or crime scenes, it is very important to locate the closest officers or emergency vehicles to the incident immediately. But how do we know where and who are the closest officers?

If every police vehicle, bicycle squad, police horse and other emergency vehicle also install a “RTK GPS system”, the control center will be able to know the exact location of officers and vehicles, therefore able to allocate the best resources to save lives.

This might save up to one minute compare to the conventional police communication method which is very valuable in life threatening or urgent situations

6.3 Recreational Users

Imagine there is a powerboat race installed with a “RTK GPS system”. Every powerboat would have a GPS receiver as well as a transmitter to transmit data to the master station. Therefore a website can be created to continuously update each powerboat position. Hence, event officials and powerboat fans will be able to follow the race closely without having to rely totally on helicopters.

Many individuals who love hiking, fishing, mountain biking, driving recreational aircraft or bush walking are interested to know their exact location, height and speed. By the use of the RTK GPS system, they will never get lost even in a foreign area again!

Furthermore, knowing the moving speed for hikers or mountain bikers will allow them to adjust their routes according to their schedule and physical strength.

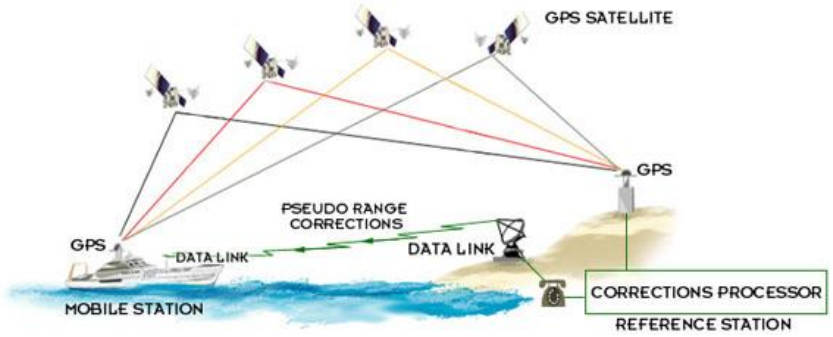
7. Differential GPS Positional over the Internet (DGPS)

DGPS is to cancel/reduce the error sources in the GPS environments due to inaccurate GPS satellite clock and orbit data, atmosphere effects as well as GPS satellite and receiver related biases.

7.1 Background to DGPS

The basic idea behind DGPS is that two (or more) receivers observing the same satellites will take similar measurements with similar errors if they are close to one another. By placing a receiver said to be a "reference receiver" at a known position, it is possible to evaluate the theoretically correct measurement values according to the known position, and then to compare these theoretical values with the actual ones taken. The difference between both values gives the measurement error, which can be used to provide corrections to receivers (mobiles) that are placed at unknown positions.

Figure 7.1 - Scenario of DGPS



For Real Time applications, a continuous data link must be established between the reference network and the rover stations in order for the DGPS users to receive the differential corrections generated from the reference network. (smartGPS, 2001)

Local and regional area differential positioning, radios and local communication systems are usually used while for wide area differential positioning satellite communication is appropriate although it is much more expensive to use. As the Internet become more and more popular, it could become a cost-effective and efficient alternative for a wide range of applications including differential satellite positioning.

7.2 DGPS Positioning over the Internet

Internet can offer many advantages over the conventional radio data transmission method when it is used for differential satellite positioning. Some of them are described in the following:

1) Internet is not limited by an effective data transmission range, therefore the rover station can be located at anywhere in earth as long as there is Internet connection available.

2) It is also advantageous for differential positioning in regions with severe signal interference such as urban areas and for applications with a large number of rover users.

The transmission time taken for the user to retrieve the differential correction data from the reference receiver station determines the differential data latency for the positioning. Less than a few seconds of data latency is typically required for most DGPS applications. To achieve this, Internet protocols should be carefully selected which defines how the data are transmitted through the Internet (i.e. TCP/IP or UDP).

7.3 The Internet Protocol best suits DGPS

The Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) are two important transport protocols that have been widely used for Internet applications. TCP provides a stream delivery and virtual connection service to applications through the use of sequenced acknowledgment with retransmission of packets while UDP provides a simple message delivery for transaction-oriented services.

TCP/IP is very reliable, but it may not suitable for high precision Real Time applications such as Real Time-Kinematic (RTK) positioning since it requires an acknowledgement to confirm the data arrival. On the other hand, UDP is able to provide faster data transmission although the reliability is not as high as TCP with a possibility of data losses.

Since fast differential data transmission is essential for the success of a Real Time positioning system to derive accurate positions, UDP protocol is preferred when dealing with the Internet-based RTK DGPS system.

8. Real Time Kinematic

RTK means the position is obtained in real time. Master station with known position while a moving rover station at unknown point. Usually RTK is considered when DGPS is being used. At least four or more satellites need to lock the position of receiver continuously in order to achieve RTK. The corrections will increase the accuracy if both master and rover stations have atmospheric biases.

8.1 Principle

The receiver should be capable of a one second update rate. The location of the reference station will be the same as for a code phase tracking DGPS system. The processor used in the reference station will compute the pseudo-range and carrier phase corrections and format the data for the communications link. (S. Wahlund., 2002)

Traditionally, both DGPS and RTK are achieved by using radio communications link, which transmits GPS correction message from master receiver to another mobile receiver. However limitations of using radio are important issues when using RTK.

Fortunately Internet seems to be a good solution to replace the radio in achieving RTK. Centmetric accuracy can be achieved if carrier measurement is transmitting in real time, and distance from master to rover station less than 20 kilometers. Possible applications of RTK such as surveying on water supply and sewer systems, car navigation and so on are benefits to our lives. True RTK can be achieved by using temporary or permanent master stations.

8.2 Temporary Master Station

A standard dual frequency receiver and a radio modem are required when using temporary master station. However, even this method can keep RTK at low cost, drawbacks such as radio range limit, additional task of setting up reference station cannot be ignored.

8.3 Permanent Master Station

When using this method, master station placing on top of the building can operate RTK 24 hours a day. It is equipped with an external radio antenna so that it provides large coverage area. Unfortunately, unless multiple users can share the cost, otherwise high cost of this manner is very hard for a single user to afford.

8.4 Network RTK

A set of reference station network is a way of increasing the range of the RTK corrections. Since the atmospheric errors are distance dependent, the single station RTK corrections may not work if distance from master to rover stations is more than 10 km. If Internet is combined with the use of RTK network, RTK service can provide high accuracy and large area coverage. (S. Wahlund, 2002)

8.5 Different Approach on DGPS and RTK

8.5.1 EUREF

EUREF (European Reference Frame) is a Sub-Commission of IAG's (International Association of Geodesy) Commission X on Global and Regional Geodetic Networks.

In order to disseminate RTCM corrections using Internet Protocol in Real Time for precise differential positioning and navigation purposes, EUREF sets up and maintains a differential GNSS infrastructure (DGNSS) on the Internet using stations of its European Permanent Network EPN. The free client software called Euref-ip-rtcm, (Windows 98/00/XP, ftp://igs.ifag.de/software/euref-ip-rtcm.zip), is available for users who want to access data streaming over Internet and connected GSM/GPRS cellular phone network for GPS positioning and other navigator applications.

This software aims to meet the growing need for Europe-wide improved Real Time determination of coordinates. It makes use of recent developments in the interconnection of mobile communication and the Internet. GPRS-based data transfer using PDAs is among the promising techniques for Real Time access to both geo-information (GIS) and geo-referencing (RTCM) data using the same communication channels.

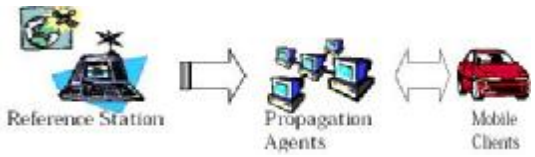
Figure 8.1 - Reference stations in EUREF

During the investigation on the protocols and other issues about Real Time GPS data streaming over Internet, for instance, latency, accuracy, system error rate, etc, few testing on this software was undertaken. However, because the list of EPN servers are only available in European countries, true RTK-GPS cannot be achieved as our rover RTK receiver was out of the maximum range from reference stations. (EUREF, 2002)

8.5.2 Differential and RTK Corrections for the Internet Car

In Japan, the common GPS correction services are depending on the radio broadcasting. Internet actually is an alternative and excellent solutions for hunting high accuracy in GPS. Internet offers many advantages over traditional radio link. First, the global Internetworking infrastructure offers high mobility and accessibility. With sufficient resources such as software, hardware, technical support, Internet can provide a low cost GPS service to people. Differs from radio communications link, radio broadcast licenses are no longer required. Wherever Internet is available, precise RTK GPS works as long as master and rover receivers are within certain acceptable range.

Figure 8.2 – Correction Information Propagation Model



The other advantage is the possibility of bi-directional communication. In this case, a request from a user will be transferred to a reference station. The reference station will supply the user with the corrections that will be formed upon his request.

These corrections will consist only of a subset of data, which help to decrease an amount of information we need to transfer through the Internet. The most important advantage of RTK via Internet is flexibility in the user requirements. It means that the user can define the accuracy of the service and consumption of the bandwidth in his request. (H. Hada, K. Uehara, H. Sunahara, 2001)

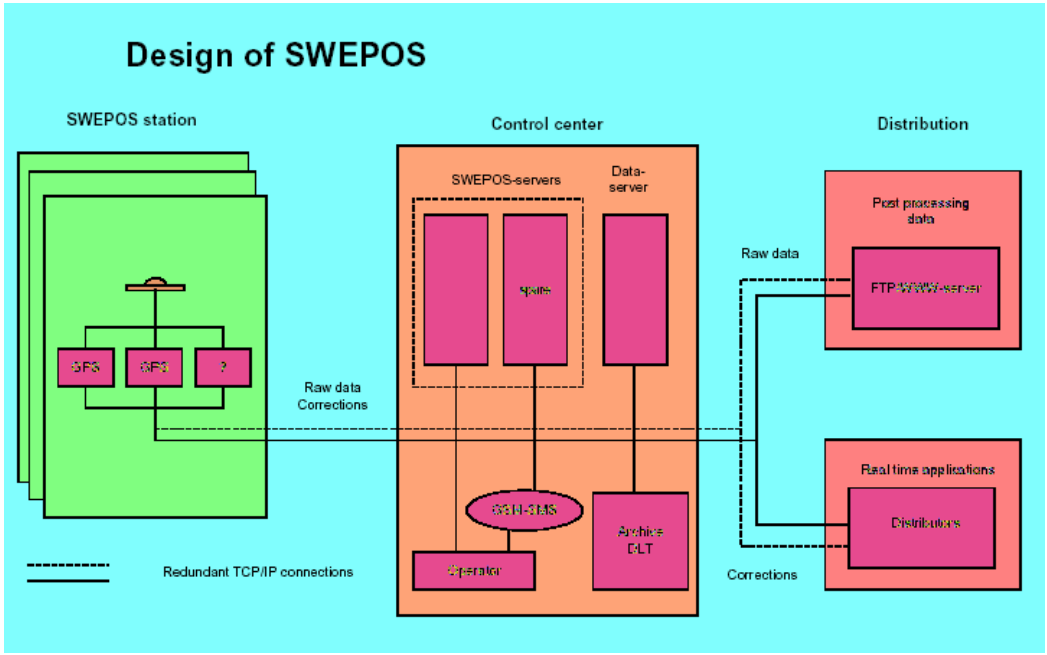
8.5.3 RTK Positioning Using Internet Protocol and GPRS in Italy

There is a permanent station connected to Internet and real time correction message transmitted with IP protocol using a Linux server through the 2101 port which is assigned by IANA. On the other hand, rover station connected to Internet receives data with a client program on Pc or PDA linked to the RTK GPS receiver.

Testing of the GPRS protocol of GSM will be arranged if telephone companies are ready. Details of this research cannot be discussed due to the lack of resources. (M. Caprioli & A. Scognamiglio, 2001)

8.5.4 SWEPOS

Figure 8.3 - Infrastructure of SWEPOS



SWEPOS is a network of permanent GPS reference stations in Sweden. It provides service about positioning in real time with meter level accuracy and by post-processing with centimeter level accuracy since 1998. One of the main purposes of SWEPO is providing DGPS and RTK data in RTCM format to real distributors. Currently all the SWEPOS stations have real time connections to the control center in Gavle through leased lines of which TCP/IP protocol is being used. Raw data and RTCM messages are transferred to control center. They are used for post-processing which can be fetched from control center through WWW/FTP-server. Then distributors of real time DGPS and RTK can receive SWEPOS real time corrections from the main control center via network of server and Internet with meter level accuracy. (G. Hedling, B. Jonsson, C. Lilje and M. Lilje, 2001)

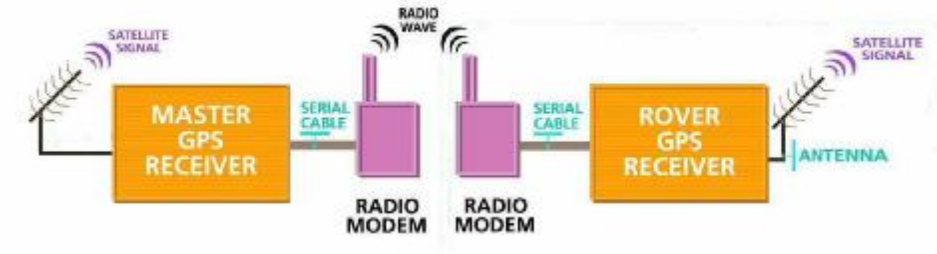
9. Communicate between GPS receivers

9.1 Use of Radio modem

Real Time Kinematic (RTK) is a high precision GPS-based positioning technique that requires raw carrier phase data (correction information) to be transmitted from a base GPS station to the mobile GPS receiver via a data link with very low latency. In order to calculate the exact position of the rover receiver, the master receiver has to continuously sending correction information to the rover receiver.

Hence the traveling time of this correction information between receivers has to be very short. Otherwise, there will be a time out error problem and the Rover receiver position can not be calculated.

Figure 9.1 - Communication between GPS receivers via radio wave

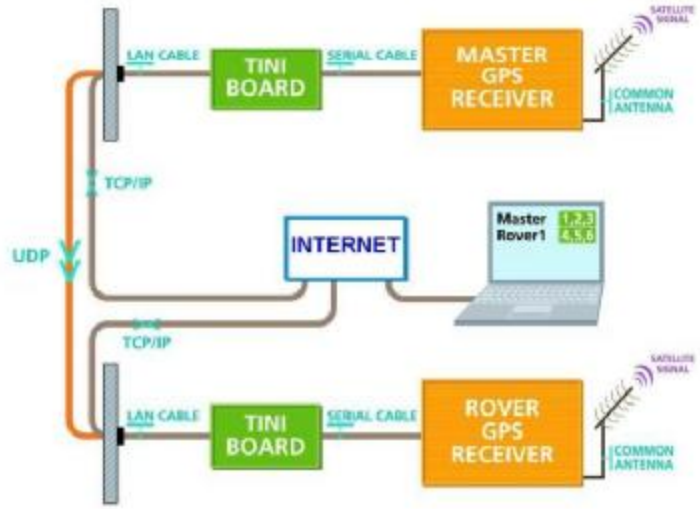


Nowadays, the most common way to send this information is using radio modems. RTCM (Radio Technical Commission Maritime) is an internationally accepted format for expressing Real Time DGPS corrections.

9.2 Converting UDP Packet

In order to send the correction message from the master receiver to the rover receiver using the Internet, these data has to convert to packets first. In this thesis project, the conversion of the data to packets is handled by the smartGPS TINI board. The size of the packet can be modified by changing the DataLog.java file, the default packet size is 512 Bytes.

Figure 9.3 - Connection between GPS receivers using TINI boards



If sending data using UDP (User Datagram Protocol), in order to choose the best packet size, the following factors have to be taken into consideration:

[Small packet size, e.g. 256 Bytes]

Advantage: Smaller packet size means the probability of there is an error in transmission would be smaller. In general, UDP packet header is small.

Disadvantage: Smaller packet size means more packets have to be sent. Since each packet has its own header, therefore more Internet traffic is generated.

[Large packet size, e.g. 1460 Bytes]

Advantage: Larger packet size can handle more data in each packet, less packets means less header information has to be wasted.

Disadvantage: Larger packet size would increase the probability of transmission errors which the data would not be resent.

But first of all, we have to make sure the message traveling time and accuracy satisfy the Real Time Kinematic requirement in the Internet environment.

Figure 9.2 - Possible location between GPS receivers

In this chapter, the Internet environment will be fully analysed via variety of experiments, including the connection speed and the packet error rate with different types of Internet protocols. Typical questions to consider are:

[Connection speed] Since the correction information required by the rover receiver is simple (ASCII data), would a 56kbps modem be capable of doing this?

[Type of connections] It is important to minimise the traveling time of the correction message and it does not really matter if there is a small amount errors for Real Time Kinematic system, would UDP be a better choice compare to TCP/IP?

[Distance between GPS receivers] Internet can be connected in most parts of the world, would that mean I can place my rover receiver anywhere as long as there is Internet connection and a clear sky is above me (to receive satellite information)?

9.3 UDP Packet Travel Time

As soon as packet arrives to the designated port, the algorithm of emulating packet switching calculates the departure time from the emulating specified link speed, delay and a size of the packet. (Creating packet switching condition for UDP packets 2002, [online])

The interval time is nothing secret. For example, if the link speed is 500kbps, then

$$\frac{\text{Speed (bits/sec)} / 8}{1{,}000{,}000} = \frac{500{,}000 / 8}{1{,}000{,}000} \text{ (byte/microsec)}$$

$$= 0.0625 \text{ byte/microsec}$$

$$= 62.5 \text{ kilo byte per second (kbps)}$$

Consider if the packet size is 512 bytes and we would like to calculate the interval time taken for each packet. We can times the packet size with a reciprocal of the calculated number above.

$$\frac{1{,}000{,}000 \times \text{bytes (bytes)}}{\text{Speed (bits/sec)} / 8} = \frac{1{,}000{,}000 \times 512}{500{,}000 / 8} \text{ (microsec)}$$

$$= 8{,}192 \text{ microsec}$$

= 0.0082 second

= around 0.01 second taken for each packet

9.4 Real Time Kinematics GPS system requirement

As mentioned in the chapter 7, typical RTK GPS requirements are:

Figure 8.4 – Summary of Real Time Kinematics GPS system requirement

Time out between Master receiver and Rover receivers	around 0.02 seconds
Physical distance between receivers	0-15 km (does not apply on this project, explained below)
Number of Satellites has to view in common	4 or 5 (sometimes)

[Time out between Master and Rover receivers]

If the time taken to send correction messages (packet) from the Master to Rover receiver is more than 0.02 seconds, the Rover information would not get enough data to compute its exact location. Therefore, our aim is to make sure the Internet packet travel time is less than 0.02 seconds.

Types of connections to consider are: 1) LAN to LAN (within a network), 2) Cable to modem and vice versa and 3) 56kbps modem to 56kbps modem. The other consideration is the transfer protocol of either TCP/IP (with respond and error checking) or UDP (without respond or error checking).

[Physical distance between receivers]

Since most people dealing with RTK GPS are using radio wave to transmit data from the master to the rover receiver, there is a separation distance limit depending on the transmission range of the radio modem. But this limitation does not apply to this thesis project, because we are using Internet as a transfer media.

[Number of Satellites has to view in common]

Although there is no radio wave limitation in this thesis project, in order to apply the Real Time Kinematics GPS technology, the master and rover receivers have to be at least able to receive four of the same satellites. Therefore, there is a physical limitation between receivers.

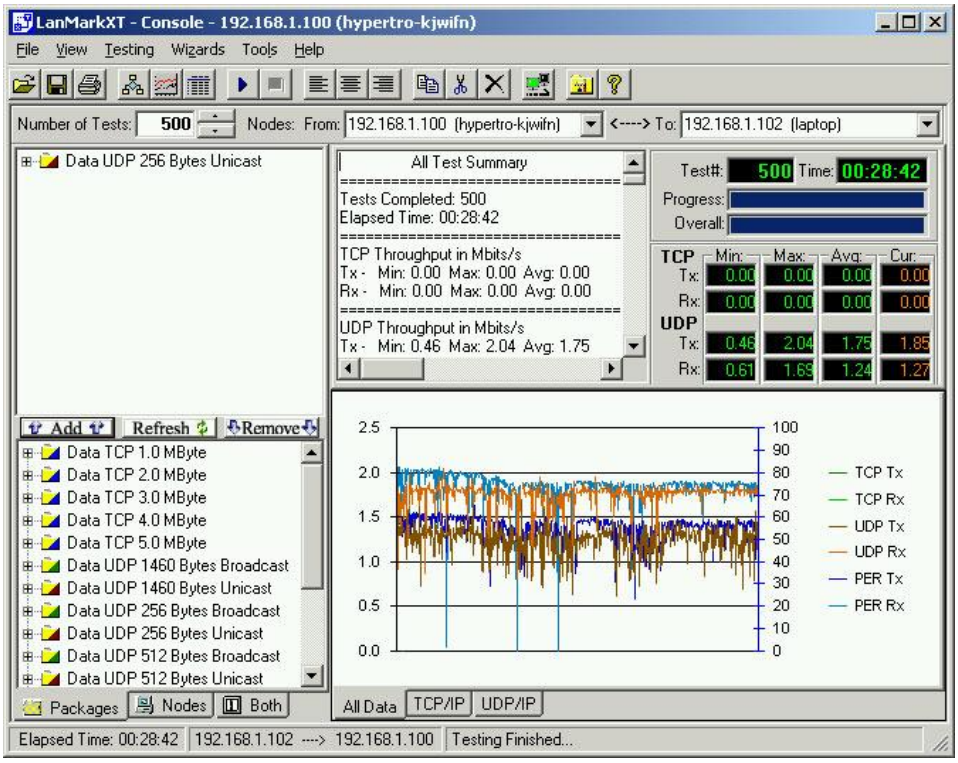
Hence, it is not practical to setup a master receiver in Sydney and a rover receiver in London, because the correction message generated by the master receiver has no use for the rover receiver, therefore the Real Time Kinematics technology cannot be used.

A practical range could be between Sydney and Newcastle for example.
Receivers can even setup at large bridges, by comparing the extremely accurate computed results, the slightly movement of the bridge can also be detected.

9.5 Analysis TCP/IP and UDP Traffic Software

In order to analysis the TCP/IP and UDP traffic, two programs were used. The first program was called LanMark XT Pro, it is a shareware with 3 days trail period at http://www.layer1software.com/products/network-performance-software/pro/

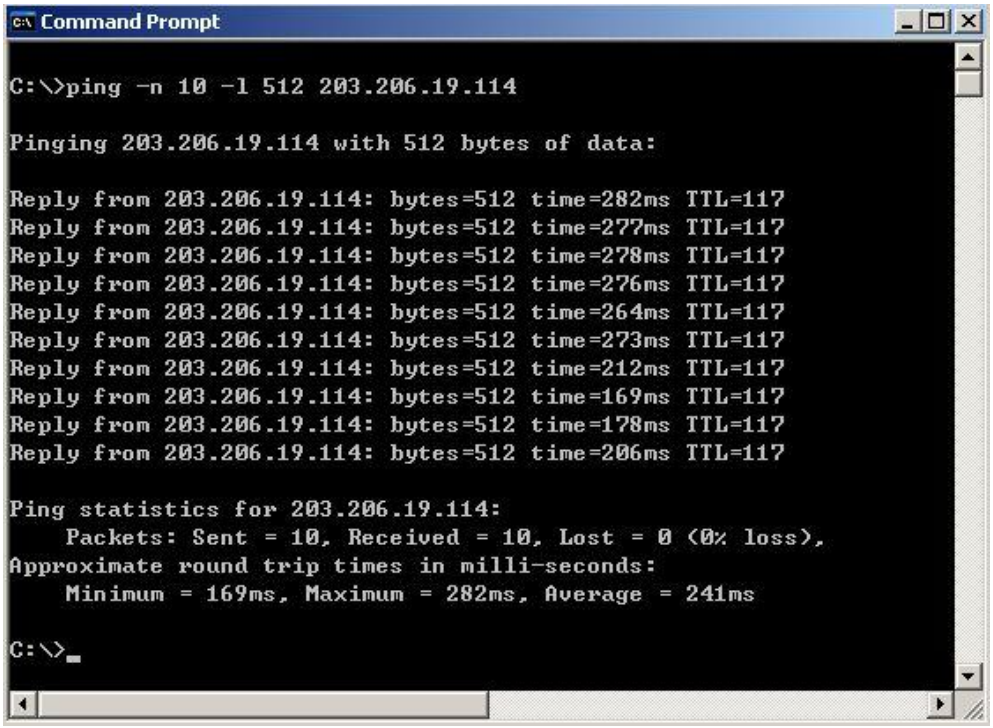
Figure 9.5 - The user interface of LanMark XT Pro



This program can be used to create real world application traffic; hence the network performance can be tested. It can generate TCP/IP or UDP traffic using different packet sizes (256, 512 or 1460 Bytes). To use this program, a folder within the program has to be shared publicly with the ability of full access control; therefore two computers can communicate and perform testing.

The Windows build-in DOS command “ping” was also used in our experiments. The PING command sends a test packet data to a designated IP address. If the data packet arrives, it will send a response and the approximate round trip times in milli-seconds could then be calculated. But this command only works with TCP/IP because UDP would not provide a response.

Figure 9.6 - The snapshot usage of the PING command


Command Prompt
C:\>ping -n 10 -l 512 203.206.19.114
Pinging 203.206.19.114 with 512 bytes of data:
Reply from 203.206.19.114: bytes=512 time=282ms TTL=117
Reply from 203.206.19.114: bytes=512 time=277ms TTL=117
Reply from 203.206.19.114: bytes=512 time=278ms TTL=117
Reply from 203.206.19.114: bytes=512 time=276ms TTL=117
Reply from 203.206.19.114: bytes=512 time=264ms TTL=117
Reply from 203.206.19.114: bytes=512 time=273ms TTL=117
Reply from 203.206.19.114: bytes=512 time=212ms TTL=117
Reply from 203.206.19.114: bytes=512 time=169ms TTL=117
Reply from 203.206.19.114: bytes=512 time=178ms TTL=117
Reply from 203.206.19.114: bytes=512 time=206ms TTL=117
Ping statistics for 203.206.19.114:
Packets: Sent = 10, Received = 10, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 169ms, Maximum = 282ms, Average = 241ms
C:\>

The additional options of the PING command that was used during testing were:

Usage:	ping	[-n count]	[-l size] target_name
Options:	-n count	Number of echo request to send.	
	-l size	Send buffer size.	
	target_name	The targeted IP address.	

Detailed test results are discussed in the section below.

9.6 Packet Travel Time Measurement

The following section analysis different Internet connection speed and Internet protocol to see whether they satisfy the requirement of Real Time Kinematics listed above. This is important because if any of these requirements not satisfied, the Rover receiver would either compute an inaccurate location or could not compute its location at all. All tests were conducted at different times of the day on Monday 16th, Tuesday 17th and Saturday 21st September 2002.

A summary of these test results is listed in the tables below. For a list of full test results, please refer to the Appendix D.

[Reminder]

$$\text{Time taken for each packet (microsec)} = \frac{1{,}000{,}000 \times \text{bytes (bytes)}}{\text{Speed (bits/sec)} / 8}$$

[For Local Area Network (LAN)]

Note: Some LAN only support up to 10Mbps. But in our experiment, the network support up to 100Mbps. But similar result would be expected with 10Mbps.

Local Area Network, UDP Unicast connection

From: UNSW, Residential college 129.94.199.101 (100Mbps)

To: UNSW, EE Blue Tooth lab 149.171.37.185 (100Mbps)

Packet size	Unicast speed	Packet travel time	Packet error rate
256 bytes	2.84 Mbps	0.7 ms	0.6%
512 bytes	3.37 Mbps	1.2 ms	2.3%
1460 bytes	5.83 Mbps	2.0 ms	6.1%

Local Area Network, TCP/IP connection

From: UNSW, Residential college 129.94.199.101 (100Mbps)

To: UNSW, EE Blue Tooth lab 149.171.37.185 (100Mbps)

Packet size	Packet travel time	Packet error rate
256 bytes	1.1 ms	0.00%
512 bytes	1.5 ms	0.00%
1460 bytes	2.6 ms	0.00%

[For a UNSW University Node to Cable modem]

University Node to Cable Modem, UDP Unicast connection

From: UNSW, Residential college 129.94.199.101 (100Mbps)

To: Optus Cable modem, Kingsford Sydney 203.164.51.563 (512kbps)

Packet size	Unicast speed	Packet travel time	Packet error rate
256 bytes	0.77 Mbps	26.5 ms	14.7%
512 bytes	0.84 Mbps	48.9 ms	42.1%
1460 bytes	1.94 Mbps	60.1 ms	50.6%

University Node to Cable modem, TCP/IP connection

From: UNSW, Residential college 129.94.199.101 (100Mbps)

To: Optus Cable modem, Kingsford Sydney 203.164.51.563 (512kbps)

Packet size	Packet travel time	Packet error rate
256 bytes	75 ms	0.00%
512 bytes	86 ms	0.00%
1460 bytes	102 ms	0.00%

[For a 56kbps modem to 56kbps modem]

56kbps modem to 56kbps modem, UDP Unicast connection

From: ihug 56kbps modem, Kensington Sydney 203.173.128.88 (33.6kbps)

To: ihug 56kbps modem, Randwick Sydney 203.173.131.70 (56.6kbps)

Packet size	Unicast speed	Packet travel time	Packet error rate
256 bytes	0.04 Mbps	45.7 ms	25.4%
512 bytes	0.07 Mbps	61.3 ms	39.2%
1460 bytes	0.15 Mbps	75.0 ms	67.1%

56kbps modem to 56kbps modem, TCP/IP connection

From: ihug 56kbps modem, Kensington Sydney 203.173.128.88 (33.6kbps)

To: ihug 56kbps modem, Randwick Sydney 203.173.131.70 (56.6kbps)

Packet size	Packet travel time	Packet error rate
256 bytes	135 ms	0.00%
512 bytes	178 ms	0.00%
1460 bytes	311 ms	0.00%

Note: The theoretical upload speed for a 56kbps modem is 33.6kbps, not 56kbps.

9.7 Packet Travel Time Analysis

Out of these 18 connections in order to select the best connection path, we need to consider the following factors:

- Packet travel time
- Connection reliability

[Packet travel time]

To select the best path as a communication link between the master and rover receiver, the packet travel time has to be less than or around 2ms (0.02 second). In the above experiments, all Local Area Network connections passed this requirement (range from 0.7 to 2.6 ms), but failed for communications which involved other networks or modem (range from 75 to 102 ms).

[Connection reliability]

There were two types of Internet protocols being examinant, TCP/IP and UDP. As expected, there is no packet error founded in any of the TCP/IP connections. However for UDP, as the size of the packet increase (from 256 to 1460 bytes) or as the connection speed becomes slower (from 100 Mbps to 56 kbps), the packet error rate increases proportionally.

For Real Time Kinematic, the packet error rate has to be less relatively small, otherwise the computed rover location would not be accurate. In the experiment performed, except for some Local Area Network UDP connections, all other UDP based connections failed this requirement.

From the experimental result, it suggested that if the connection involved traffic outside a local area network (for example involving other network or modem users), then the transfer of packet (correction information) between receivers would not be fast enough to satisfy the Real Time Kinematic requirement.

Hence, only LAN connections can satisfy the RTK GPS requirements:

From (speed)	To (speed)	Type of connection	Packet size (bytes)	Travel time (micro-sec)	Error rate (%)
100MbpsLAN	100MbpsLAN	UDP	256 bytes	0.7	0.11%
100MbpsLAN	100MbpsLAN	UDP	512 bytes	1.1	2.67%
100MbpsLAN	100MbpsLAN	UDP	1460 bytes	2.0	6.11%
100MbpsLAN	100MbpsLAN	TCP/IP	256 bytes	1.1	0%
100MbpsLAN	100MbpsLAN	TCP/IP	512 bytes	1.5	0%
100MbpsLAN	100MbpsLAN	TCP/IP	1460 bytes	2.6	0%

From these six connections, in order to select the best connection, a decision has to be made on whether TCP/IP or UDP is to be used. For Real Time Kinematic, it is best to use UDP since the packet travel time is relatively less than the TCP/IP. Although there would be error packet generated when using UDP, it will not affect the accuracy of the computed position if the error rate is relatively small.

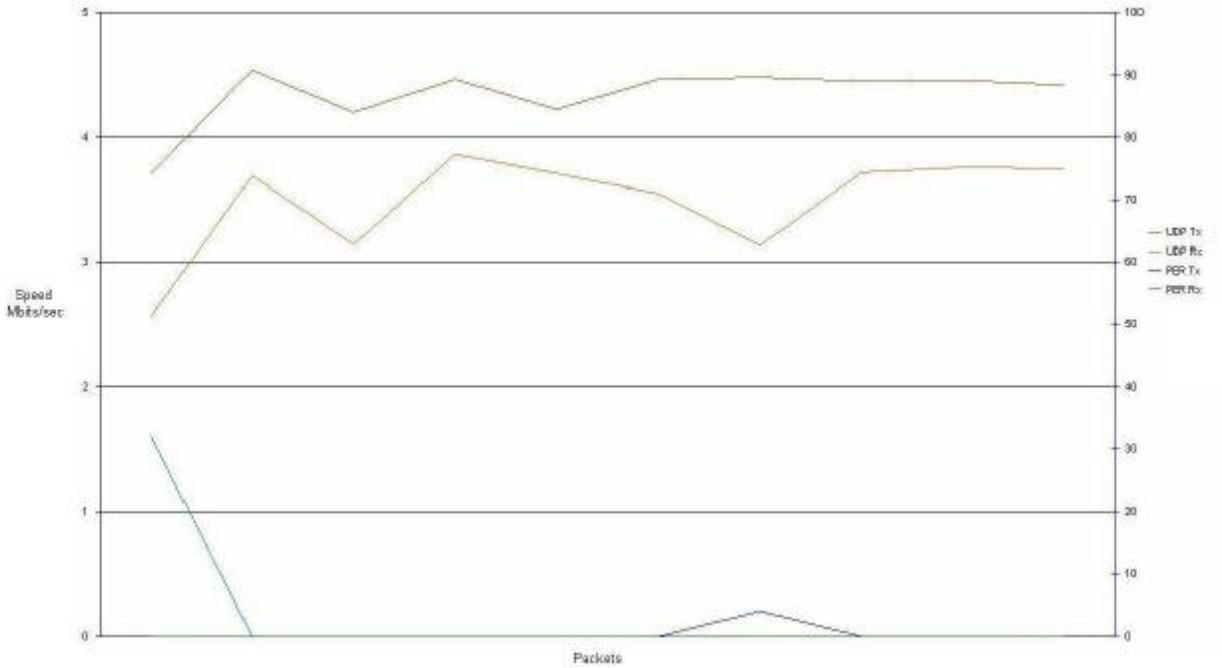
Out of the three UDP LAN connections, since the connection with 1460 bytes packet size has a relatively large packet travel time and packet error rate, so it is not a good choice. On the other hand, the connections with LAN UDP 256 bytes and LAN UDP 512 bytes also fully satisfy the Real Time Kinematic requirement.

When a Local Area Network is not fully reliable (for example there are other network traffics), it is better to use 256 bytes as the packet size, because its packet error rate is smaller. Otherwise, LAN UDP with 512 bytes packet size is the best choice for RTK GPS out of all examined connections.

A summary of the best selected connection link between Master and Rover receivers' network performance supporting Real Time Kinematic is listed below:

Optimised Settings	
From (speed):	100 Mbps LAN
To (speed):	100 Mbps LAN
Type of connection:	UDP
Experimental Result	
Packet size:	512 bytes
Unicast speed	3.37Mbps
Packet travel time:	1.2 micro-sec
Packet error rate:	2.3%

Figure 9.7 - The LAN UDP (512bytes) Connection speed

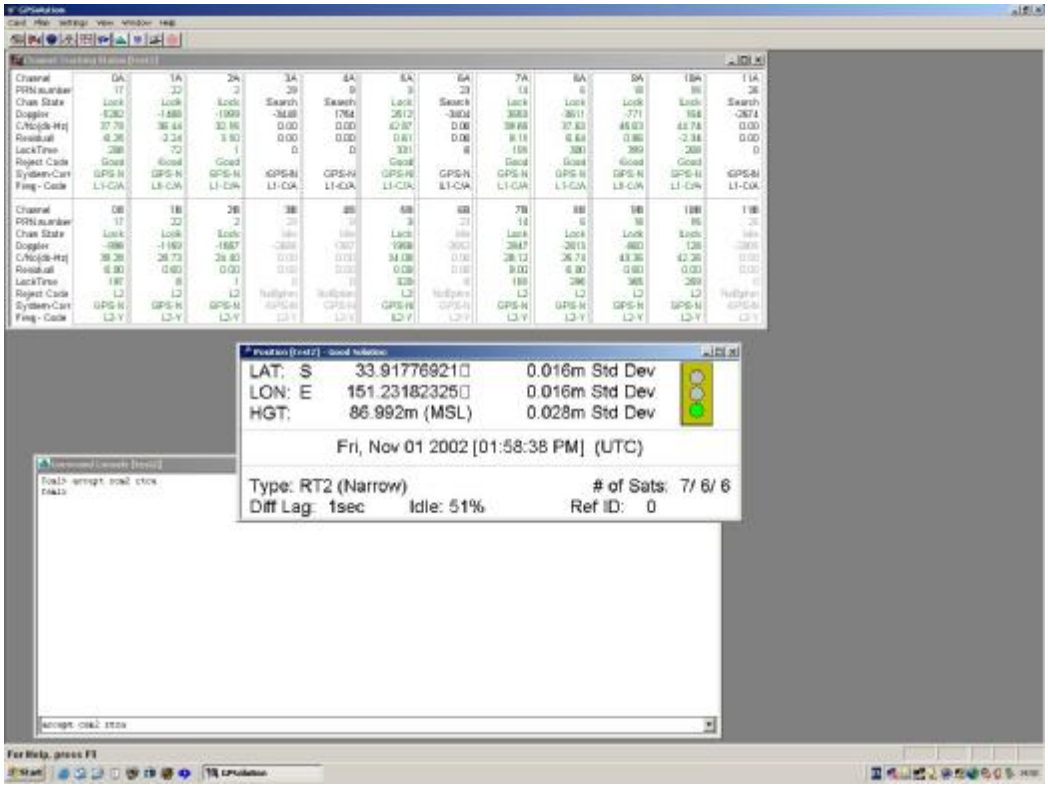


10. GPS Applet Controller

10.1 GPSolution

The Master and Rover receiver that was used in this project belongs to a package called MiLLennium GPSCard which was manufactured by NovAtel Inc. located in Canada. The package comes along with a software called GPSolution which is a Microsoft Windows-based graphical user interface that allow users to access the receivers' features.

Figure 10.1 - A graphical user interface (GPSolution) for the GPSCard



But if the user is not using Microsoft Windows (such as Linux) or have not got this program installed in the computer, then he or she would not be able to control the Master or Rover receivers.

10.2 TINI LCD Screen

Last year, the smartGPS group created a LCD screen with input buttons attached to the TINI board. This LCD screen can display up to 80 characters in 4 lines while the three input buttons can perform selections like up, down and enter in the smartGPS menu.

Figure 10.2 - smartGPS TINI LCD screen with input buttons

The three programmed commands included:

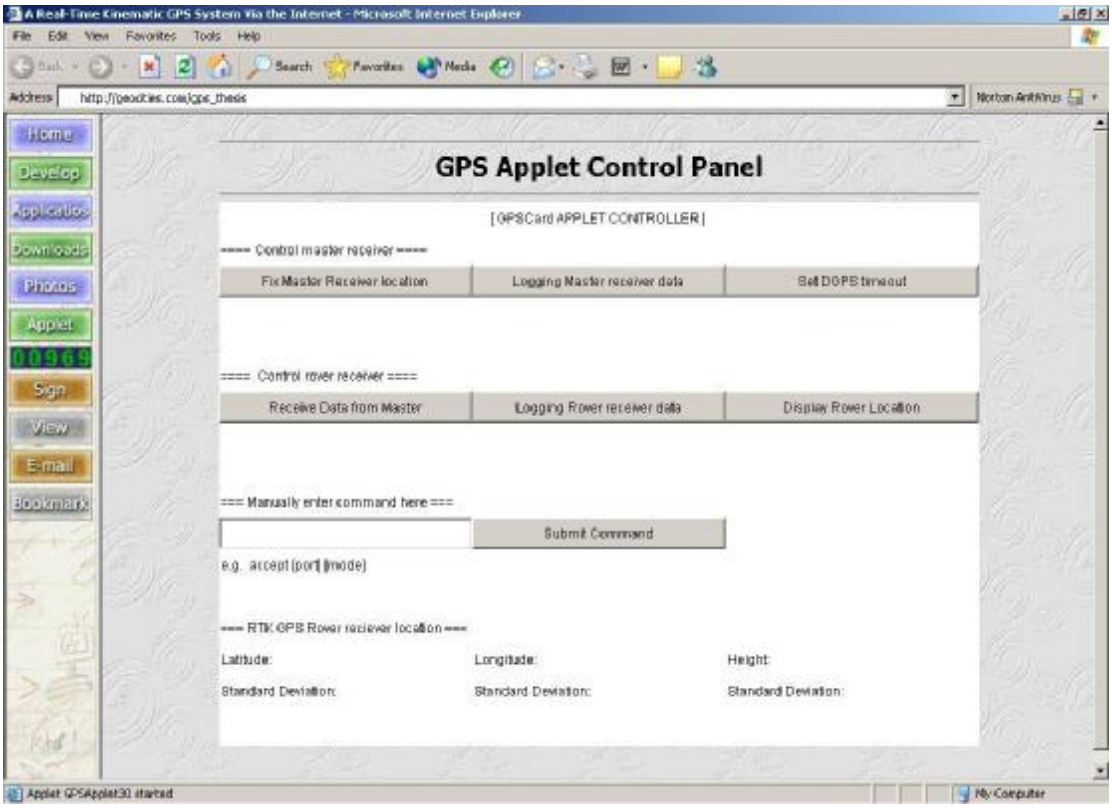
- Select the correct Baud rate (4800, 9600 or 19200)
- Logging GPS Data (to either Memory card or Remote PC)
- Control the brightness of the LCD screen

Although this is a very user friendly design, it can only select the three commands (out of sixty two possible commands – listed in the appendix) which are already programmed in the TINI board. Hence, many advance instructions for the receivers can not be used.

10.3 GPS Applet Controller

To overcome the above problems, a Java applet called “GPSApplet[version].java” is built. It can be opened using the file “GPSApplet.html”, these files are attached in Appendix C and the CD. This Java applet can be uploaded to a homepage. Hence, as long as there is a computer which supports Java and has Internet connection, it can be used to control the master and rover receivers.

Figure 10.3 - GPS Applet Control Panel



To use this applet, it is assumed that the computer’s serial port COM1 is connected to the receiver’s COM1 port, and that a remote terminal is connected to the receiver’s COM2 port. Also, this computer is required to connect to the Internet or a local area network.

10.4 Communicate with the Receiver

Communication with the receiver consists of issuing commands through the COM1 or COM2 port from an external serial communications device. The communication between the GPS receiver and a computer is achieved by virtue of the GPS firmware that resides within the GPS receiver.

For communication to occur, both the receiver and operator interface have to be configured properly. The receiver's default port settings are as follows:

- RS232C
- no parity
- 1 stop bit
- echo off
- 9600 bps
- 8 data bits
- no handshaking

Changing the default settings requires using the COMn command, which is described in the MiLLennium Command Descriptions Manual.

Although the receiver can operate at bit rates as low as 300bps, this may not always be desirable. For example, if several data logs are active. (i.e. a significant amount of information needs to be transmitted every second) but the bit rate is set too low, data will overflow the serial port buffers and cause an error condition in the receiver status.

10.4.1 Fix Master Receiver Location

The reference station must initialize the precise position of its reference antenna phase center (lat/lon/hgt). This is accomplished by utilizing the GPSCard FIX POSITION command. The fixed position in the following command is the antenna located at the rooftop of the EE building, UNSW.

Command: FIX POSITION [lat] [lon] [height]

Applet 檢視器 : GPSApplet30.class
Applet
[GPSCard APPLET CONTROLLER]
==== Control master receiver ====
Fix Master Receiver location
Logging Master Receiver data
Set DGPS timeout
Submitted: fix position -33.9 -115.2 86.9

10.4.2 Logging Master Receiver data

LOG stores or displays data on a communication port (COM1 or COM2). If the LOG syntax does not include a trigger type, it will be output only once following execution of the LOG command. If trigger type is specified in the LOG syntax, the log will continue to be output based on the trigger specification. Specific logs can be disabled using the UNLOG command, whereas all enabled logs will be disabled by using the UNLOGALL command. All activated logs will be listed in the receiver configuration status log (RCCA).

Command: LOG [port] [data] [ontime] [seconds]

Applet 檢視器 : GPSApplet30.class
Applet
[GPSCard APPLET CONTROLLER]
==== Control master receiver ====
Fix Master Receiver location
Logging Master Receiver data
Set DGPS timeout
Submitted: log com2 RTCAREF ontime 10

Note: RTCA = Standard Format Data Logs (Binary)

10.4.3 Set DGPS Timeout

The DGPSTIMEOUT command allows the reference station to set the delay by which it will inhibit utilization of new ephemeris data in its differential corrections. This delay ensures that the remote receivers have had sufficient time to collect updated ephemeris data as well.

Command: DGPSTIMEOUT [dgps delay] [ephem delay]

Since this thesis project is dealing with Real Time Kinematics, therefore only 2 seconds of delay is given to rover stations to collect updated ephemeris. After the delay period is passed, the reference station will begin using new ephemeris data.

10.4.4 Receive Data from Master

This command is used to set the GPSCard's COM port command interpreter for acceptance of various data formats. Once initialized, the rover GPSCard receiver will operate in single point mode until the differential message are received. If data is lost, the GPSCard will revert to single point positioning until the pseudorange correction message are restored.

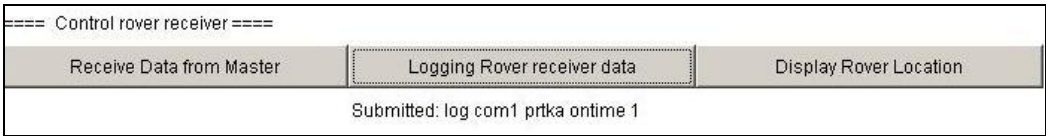
Command: ACCEPT [port] [mode]

==== Control rover receiver ====
Receive Data from Master
Logging Rover receiver data
Display Rover Location
Submit: accept com2 rtcm

10.4.5 Logging Rover Receiver Data

The LOG function was explained in two pages before.

Command: LOG [port] [data] [ontime] [seconds]

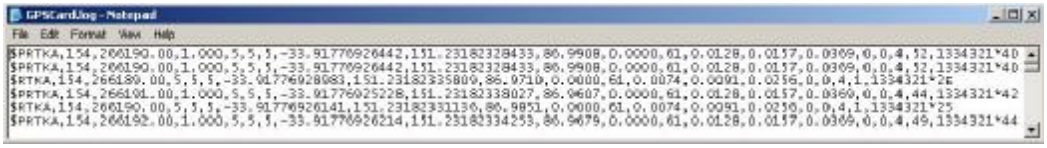

==== Control rover receiver ====
Receive Data from Master
Logging Rover receiver data
Display Rover Location
Submitted: log com1 prtka ontime 1

Note for PRTKA: Type = Positioning, Trigger = ontime or onmark

10.4.6 Display Rover Location

When this button is pressed, the Rover location would be displayed in the table at the bottom of the applet. It gets the data from the Rover receiver:

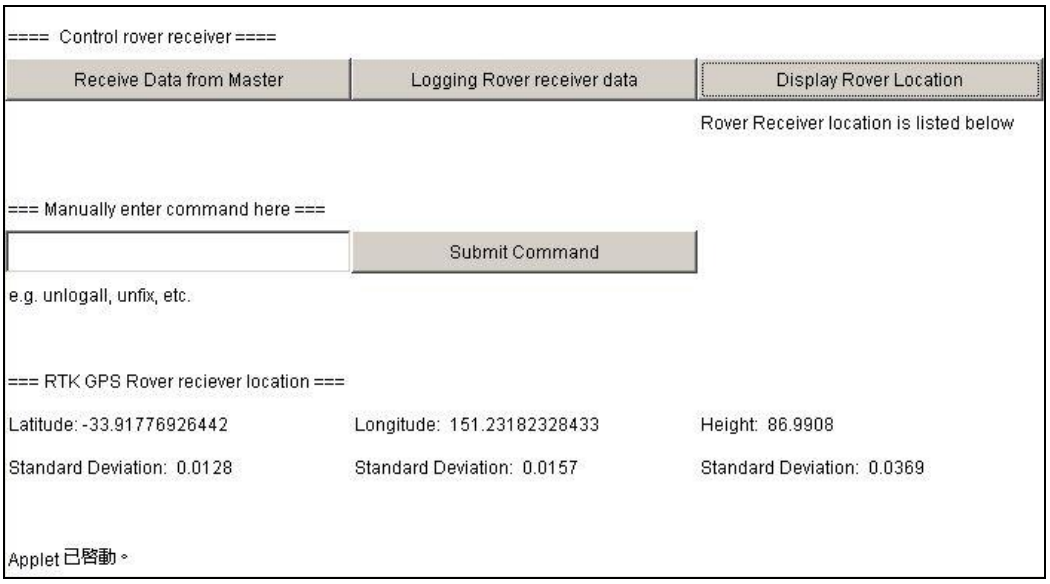
Figure 10.4 - A sample of the logged data from the Rover receiver



By using the Java Communication package (javax.comm), the RS-232 serial communications port which is a low-level made available by the underlying system. SerialPort defines the required functionality for serial communication ports. This applet reads the logged data from the serial port and converts them into type String. By using array, it then selects the useful information and displays them in the table.

There are two ways this button can work:

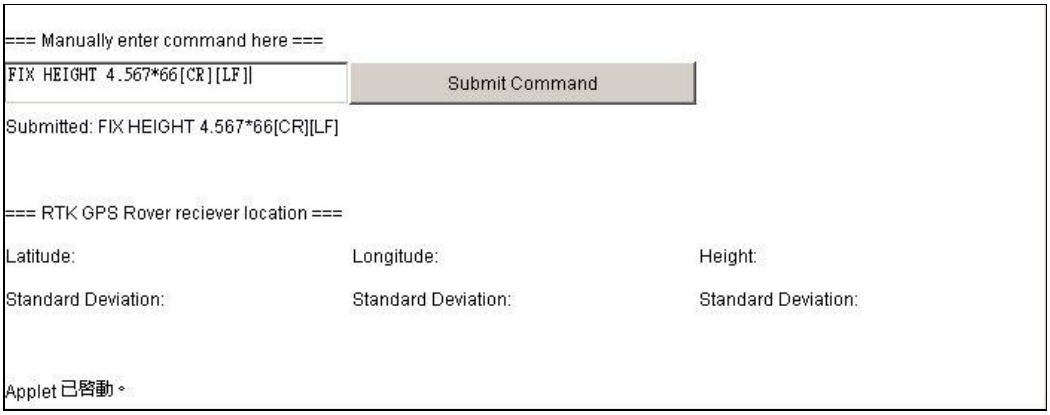
- Displaying logged data from COM1 or COM2 in real time
- Displaying logged data from a file which has been saved



10.4.7 Manually enter command

Besides the six most frequent used commands above, there are a total of sixty two possible commands which can be used by the receiver. To use these commands, users can simply enter the command in the textbox provided in the applet. A confirm message will be displayed once the command has been submitted.

The next three frequently used commands are: [unlog], [unfix] and [fix height]. For a listed of the full sixty two commands, please refer to the appendix



10.5 Editing the GPSApplet Controller

Modification on the java applet can be made on file GPSApplet[version].java Applet can be opened by running the file GPSApplet.html using a browser. But since the browser has cache memory, therefore even modifications are made on the applet, the browser usually can be updated promptly.

To overcome this problem, the “appletviewer” which comes with the Java development kit can be used. In the DOS command prompt, change to the directory which contains the compiled GPSApplet[version].java and GPSApplet.html files and type: appletviewer GPSApplet.html.

10.6 Accuracy of the Computed Locations

After the Rover receiver computed its location, for example (longitude 123, latitude 456 and height 789), it is hard to know how accurate these values are unless these values come with the standard deviation. A good example of a computed Rover location using Real Time Kinematics is:

Figure 10.5 - Comparison of the Actual and Rover receiver computed values located at EE Building, UNSW, Sydney

	Actual location	Computed Values	Standard deviation
Latitude	-33.91776927	-33.91776529	0.011
Longitude	151.2318233055	151.23182334	0.015
Height	86.983	86.995	0.026

The above computed results are extremely accurate, as the standard deviations are only around 2 cm (0.02 meter). This is because the above value was taken when the master and rover were using the same antenna located at the rooftop of the EE building with the communication using 100Mbps within the University network.

If the communication link between the receivers is not fast enough (packet travel time is long) or not accurate enough (packet error rate is large), it will directly affect the accuracy of the computed Rover receiver position.

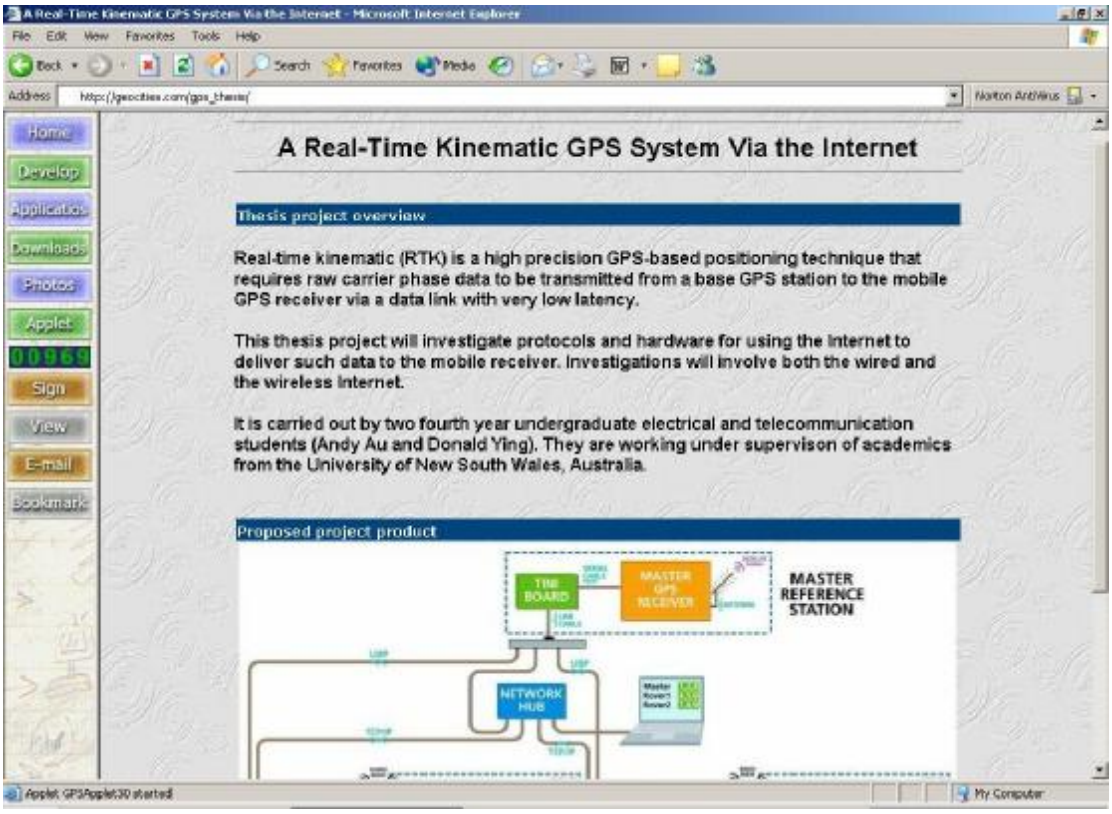
11. Thesis Project Promotion

11.1 Website

In order to share the achievements of the thesis project with other research fellows and future students, a website is created and located at:

http://geocities.com/gps_thesis/

Figure 11.1 – A Real Time Kinematic GPS System via the Internet Website


A Real-Time Kinematic GPS System Via the Internet
Thesis project overview
Real-time kinematic (RTK) is a high precision GPS-based positioning technique that requires raw carrier phase data to be transmitted from a base GPS station to the mobile GPS receiver via a data link with very low latency.
This thesis project will investigate protocols and hardware for using the Internet to deliver such data to the mobile receiver. Investigations will involve both the wired and the wireless Internet.
It is carried out by two fourth year undergraduate electrical and telecommunication students (Andy Au and Donald Ying). They are working under supervison of academics from the University of New South Wales, Australia.
Proposed project product
MASTER REFERENCE STATION
TIME BOARD
NETWORK HUB
Home
Develop
Applications
Downloads
Photos
Applet
Sign
View
E-mail
Bookmark

The website contains six sub-pages with a guest book.

Home: General description of the thesis project.

Develop: Contain the full development plan (step by step)

Challenges and possible risks.

Application: Listed the three possible market applications

Courier service, Emergency service allocation & Recreational users

Download: Project proposal, Final report, Source code files and GPSCard menu can be downloaded here. Useful links to other website are also provided.

Photos: A photo collection of the equipment used in the project with brief descriptions.

Applet: The GPSApplet Controller is put here, but it cannot be used to control the Master or Rover receiver unless all the settings required are correct.

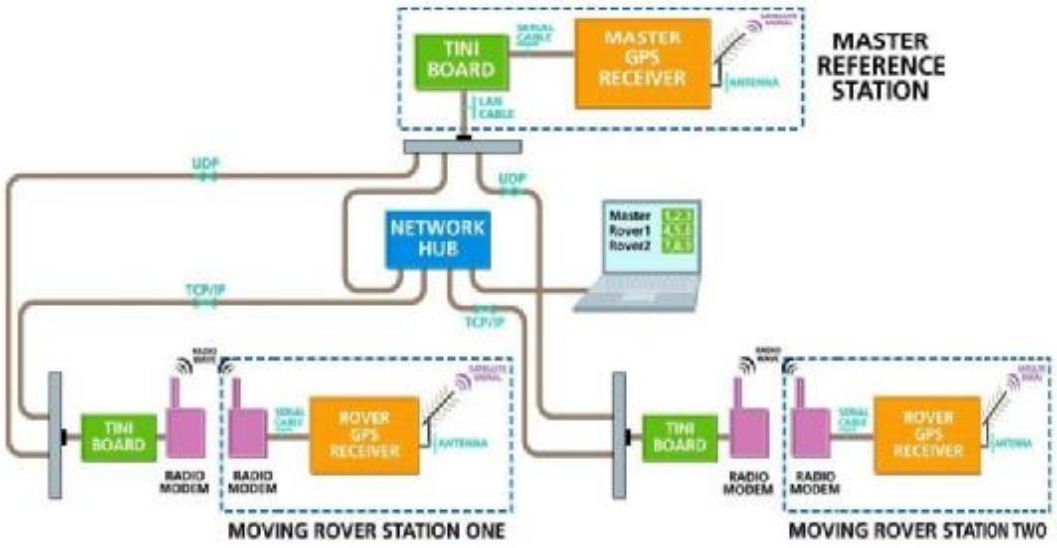
11.2 Promotional CD

A CD has been produced and attached with this report; it contains the proposal, final report, all the software developed and the user manual of the GPSCard. This should be very useful for the people who are interested at our project, especially future students.

12. Further Development

The aim of the system is to provide accurate rover GPS receiver locations by using Real Time Kinematic technology. Looking at the system diagram below, there is nothing can be changed on the GPS receivers, radio modems and the Internet protocols (TCP/IP or UDP). But there are some further developments on the TINI board and the method of handling GPS receiver data that would further enhance the attractiveness of the project.

Figure 12.1 - A Real Time Kinematic GPS system via the Internet



12.1 Additional Serial ports and Address space

The address space of the TINI is designed in such a way that the user has complete control and freedom to add extra devices. Hence, logic to provide additional I/O can be added in order to interface additional serial ports and other devices.

When broken down, serial ports consist of a two wire asynchronous serial interface. One line is needed for transmitting, the other for receiving. Other common serial port handshaking lines such as CTS and RTS could also be added, but most GPS receivers do not require them for use. (smartGPS, 2001)

Any number of serial ports could be added to a smartGPS unit with only minimal hardware and software routines. These software routines would have to be written in native assembler code as Java would not be fast enough for this asynchronous communication.

12.2 Host Internet Server

A web server can be hosted at the University domain with the applet controller; hence the system can be used in areas around the University. Also, people around the world can monitor the rover receiver position as long as they have Internet access.

12.3 Display Features

The TINI LCD display interface can be easily updated or altered. This will allow for additional smartGPS features in further development. (smartGPS, 2001)

12.4 Further Experimentation

Experiments with further GPS technology, for example carrier phase tracking methods, would be advisable in order to make the Real Time Kinematic System a more accurate package.

12.5 Remove Unnecessary Data before Transmitting

To increase the efficiency of the computed rover receiver results, the packet travel time from master to rover receiver has to be as least as possible. At the moment, the master receiver is sending a large amount of duplicated data:

```
Structure: [$PRTKA] [week] [sec] [lag] [#sv] [#high] [L1L2 #high] [lat] [lon] [hgt]
           [undulation] [datum ID] [lat SdDev] [lon SdDev] [hgt SdDev] [soln status]
           [rtk status] [posn type] [idle] [stn ID] [*xx] [ [CR][LF] ]

           $PRTKA,154,266210.00,1.000,6,5,5,-33.91776928615,151.23182333583,86.9865
           ,0.0000,61,0.0128,0.0157,0.0369,0,0,4,54,1334321*47

           $PRTKA,154,266211.00,1.000,6,5,5,-33.91776926423,151.23182332702,86.9862
           ,0.0000,61,0.0128,0.0157,0.0369,0,0,4,50,1334321*46
```

The text in red is the duplicated data. Out of the 123 characters in each line, 51 characters are redundant and occupy 41% of bandwidth.

If only relevant data is sent, only 73 of the 123 characters are required, hence the packet propagation time to receivers can be greatly reduced. The removed data will be added back into the instruction stream at the receiver before passing it onto the hardware. The process of adding data would take much less time than actually transmitting them.

At the moment, the packet size is 512 bytes; therefore around 4 lines (123 characters each) of data can be fitted in each packet. But if each line is reduced to 73 characters, then each packet can fill at least 7 lines of data.

12.6 AppleTalk

According to the experiment performed, the local area network is the best transmission path between GPS receivers. In fact, beside TCP/IP and UDP, Apple Mac computers also support AppleTalk as a data transfer protocol.

AppleTalk Update-Based Routing Protocol (AURP) is a tunneling and a routing protocol. AURP provides the feature of AppleTalk tunneling in TCP/IP. This feature enables two isolated AppleTalk networks to be connected by way of a TCP/IP network. AURP provides update-based routing and reliable delivery of routing information. To reduce the amount of bandwidth, update-based routing sends updates to peer routers only when network routing information changes, rather than sending periodic broadcasts of the routing table.

Therefore, having an AppleTalk implementation in our program (instead of creating TCP/IP packets manually) could be an easier way to get the program running an AppleTalk only network environment.

13. Conclusion

Based on the experimental results, most TCP/IP or UDP transmission with different packet size within a Local Area Network are also capable of providing throughput fast enough to accommodate Real Time Kinematic GPS applications.

Among these connections, the best transmission path between the Master and Rover receiver is using the UDP protocol with 512 bytes packet size within a Local Area Network (10/100Mbps). On average, it only requires 1.2 micro-sec to transfer a packet with 2.3% packet error rate.

It has also been demonstrated that the use of Cable modem (512kbps) or 56kbps modem does not satisfies the RTK latency minimum requirement, although some Cable modem connections are suitable for other GPS strategies such as DGPS.

The GPSApplet Controller provides a costless and advantageous way to control the Master or Rover receiver in any operating system. Further development suggestions have been made so that the Real Time Kinematic can further reduce the packet travel time between GPS receivers.

14. References

This thesis project is built upon the previous year's project - smartGPS
Written By: Adrian Clark, Antony Histon, Peter von Konigsmark
Electrical Engineering Undergraduate Project, UNSW (2001)

Australian Communications Authority, Class Licensing, Short Range Spread Spectrum Devices, 2001 [online]
http://www.aca.gov.au/publications/info/spreadsp.htm

Bertsekas D. & Gallager R., Data Network, Prentice-Hall, 2nd Edition, 1992

Chancogne D. & Austin M., A short Java course, 1996 [online]
http://www.isr.umd.edu/~doc/Java/Course/chap3.html

Corvallis Mcirotechnology lnc., Introduction to GPS, 2002 [online]
http://www.ctinc.com/gpsbook/index.htm

Dick Smith Electronics - Product Catalogue 2001-2002 [online]
http://www.dse.com.au

DHL Australia: Tracking center, May 2002 [online]
http://www.dhl.com.au

Goldman J., Rawles P., Applied Data Communications – A Business-Oriented Approach, 3rd Edition, 2001

GPS/GLONASS Data Center, Germany, 2002 [online]
http://igs.ifag.de/index_euref.htm

Gran D. M., An Introduction to Java, 2001 [online]
http://www.cs.bsu.edu/~bsjug/intro.html

Hada H., Uehara K., Sunahara H., Murai J., Petrovski I., Torimoto H., Kawaguchi S., Differential and RTK corrections for the Internet car, 1999 [online] http://neo.sfc.wide.ad.jp/~watari/WEB/gnss99.pdf

Hedling G., Jonsson B., Lilje C. & Lilje M., SWEPOS – The Swedish Network of Permanent GPS Reference Station, June 2001

Huber J. F., Weiler D. & Brand H. UMTS, the Mobile Multimedia Vision for IMT-2000: A Focus on Standardization. IEEE Communications Magazine, September 2000

Kurose J. & Ross K., Computer Networking - A Top-Down Approach Featuring the Internet, Addition Wesley, 1st Ed., 2001

Lachapelle R,. & Fattouche M., Cellular Telephone Positioning Using GPS Time Synchronization. GPS World, April 1998

Loomis D., The TINI Specification and Developer's Guide. New York: Addison-Wesley, 2001

McElroy S., Robins I., Jones G. & Kinyside D., Exploring GPS – A GPS users guide, 2nd Edition, 2001

MiLLennium GPSCard and Enclosures – Guide to Installation and Operation, 2001 [online]
http://www.novatel.ca

Muellerschoen R. J., Bertiger W. I., & Lough M.F., Results of an Internet-Based Dual-Frequency Global Differential GPS System. Procedings of IAIN World Congress, San Diego, CA, June 2000

National Air and Space Museum - How does GPS works, May 2002 [online] http://www.nasm.si.edu/galleries/gps/work.html

Network-Based Techniques for RTK Applications, 2001 [online]
http://www.ion.org/meetings/past/gps2001/d1.html
Real Time Kinematic differential GPS surveys, 2001[online]
http://www.nap.usace.army.mil/channel/em/16.pdf

SNAP, Australia, 2002 [online]
http://www.gmat.unsw.edu.au/snap

Shields S., Flinn J. & Obregon A, GPS in Pits: Differential GPS Applications at Moreni Copper Mines. GPS World, 2000

Tanenbaum A. S., Computer Networks, Prentice-Hall, 3rd Ed., 1996

Thai B., Undergraduate Course TELE3018 Data Network 1 [online]
http://alpha400.ee.unsw.edu.au/~tele3018

Thales Company Technologies: Differential GPS, May 2002 [online]
http://pro.thalesnavigation.com/us/pro/scripts/techno_gps/techno_gps_dgps.asp

TINI – ibutton.com, May 2002 [online]
http://www.ibutton.com/TINI/

Tsukamoto A., Creating packet switching condition for UDP packets, 2002 [online]
http://www.columbia.edu/~at541/src/emulator.html#Algorithm

Trimble Navigation Limited, 2002 [online]
http://www.trimble.com/gps

Tyson J., How OSI works, 2001 [online]
http://www.howstuffworks.com/osi.htm

Wahlund S., Production measurements with Network RTK-Tests and analysis, 2002

Weber G., Real Time Streaming of DGPS Corrections over Internet, June 2002

Wilson G., OSI layer model, 2001 [online]
http://www.geocities.com/SiliconValley/Monitor/3131/ne/osimodel.html

Wormley S. J., Sam Womley's DGPS, 2002 [online]
http://www.edu-observatory.org/gps/dgps/htm

Yang C. & Kim S., The current status of GPs network, datum transformation and real time kinematic GPS positioning in Korea, 2001

15. Appendix

Appendix A - Standard Command Tables

Table 2-1 Commands By Function Table

COMMUNICATIONS, CONTROL AND STATUS	
Commands	**Descriptions**
ANTENNAPOWER	Power to the low-noise amplifier of an active antenna
COMn	COMn port configuration control
COMn_DTR	DTR handshaking control
COMn_RTS	RTS handshaking control
DIFF_PROTOCOL [1]	Differential Protocol Control
FREQUENCY_OUT	Variable frequency output (programmable)
LOG	Logging control
MESSAGES	Disable error reporting from command interpreter
RINEX	Configure the user defined fields in the file header
RTCMRULE	Sets up RTCM bit rule
RTCM16T	Enters an ASCII message
SEND	Sends ASCII message to COM port
SENDHEX	Sends non-printable characters
SETL1OFFSET [1]	Add an offset to the L1 pseudorange to compensate for signal delays

1 Intended for advanced users of GPS only

GENERAL RECEIVER CONTROL AND STATUS	
Commands	**Descriptions**
$ALMA	Download almanac data file
CRESET	Reset receiver to factory default
DYNAMICS	Set correlator tracking bandwidth
HELP	On-line command help
RESET	Performs a hardware reset (OEM only)
SAVEALMA	Saves the latest almanac in NVM
SAVECONFIG	Saves current configuration (OEM only)
$TM1A	Injects receiver time of 1PPS
VERSION	Software/hardware information

POSITION, PARAMETERS, AND SOLUTION FILTERING CONTROL	
Commands	**Descriptions**
CSMOOTH [1]	Sets amount of carrier smoothing
DATUM	Choose a DATUM name type
ECUTOFF	Satellite elevation cut-off for solutions
FIX HEIGHT	Constrains to fixed height (2D mode)
FIX POSITION	Constrains to fixed lat, lon, height
FRESET	Clears all data which is stored in NVM
$IONA	Download ionospheric correction data
IONOMODEL	What ionospheric correction to use (MiLLennium with the WAAS option)
LOCKOUT	Deweights a satellite in solutions
$PVAA [1]	Position, velocity and acceleration in ECEF coordinates
RTKMODE	Setup the RTK mode
UNDULATION	Ellipsoid-geoid separation
USERDATUM	User-customized datum
WAASCORRECTION	Controls handling of WAAS corrections.

1 Intended for advanced users of GPS only.

SATELLITE TRACKING AND CHANNEL CONTROL	
Commands	**Descriptions**
$ALMA	Download almanac data file
ASSIGN	Satellite channel assignment
CONFIG	Switches the channel configuration of the GPSCard
DYNAMICS	Sets correlator tracking bandwidth
FIX VELOCITY	Aids high velocity reacquisition
RESETHEALTH	Reset PRN health
SETHEALTH	Overrides broadcast satellite health

WAYPOINT NAVIGATION	
Commands	**Descriptions**
MAGVAR	Magnetic variation correction
SETNAV	Waypoint input

DIFFERENTIAL REFERENCE STATION	
Commands	**Descriptions**
DGPSTIMEOUT	Sets ephemeris delay
FIX POSITION	Constrain to fixed (reference)
LOG	Selects required differential-output log
POSAVE	Implements position averaging for reference station
RTCMRULE	Selects RTCM bit rule
SETDGPSID	Set reference station ID

DIFFERENTIAL REMOTE STATION	
Commands	**Descriptions**
ACCEPT	Accepts RTCM1, RTCA or RTCAB differential inputs
$ALMA	Input almanac data
DGPSTIMEOUT	Set maximum age of differential data accepted
RESET	Performs a hardware reset
$RTCA	RTCA differential correction input (ASCII)
$RTCM	RTCM differential correction input (ASCII)
RTCMRULE	Selects RTCM bit rule
SETDGPSID	Select differential reference station ID to receive

CLOCK INFORMATION, STATUS, AND TIME	
Commands	**Descriptions**
CLOCKADJUST	Enable clock modelling & 1PPS adjust
DIFF_PROTOCOL [1]	Differential protocol control
EXTERNALCLOCK	Sets default parameters of an optional external oscillator
EXTERNALCLOCK FREQUENCY	Sets clock rate
SETTIMESYNC [1]	Enable or disable time synchronization
$UTCA	Download UTC data

1 Intended for advanced users of GPS only

Table 2-2 GPSCard Command Summary

Command	Description	Syntax
$ALMA	Injects almanac	(follows NovAtel ASCII log format)
$ION A	Injects ionospheric refraction corrections	(follows NovAtel ASCII log format)
$PVAA	Injects latest computed position, velocity and acceleration	(follows NovAtel ASCII log format)
$REPA	Injects raw GPS ephemeris data	(follows NovAtel ASCII log format)
$RTCA	Injects RTCA format DGPS corrections in ASCII (Type 1)	(follows NovAtel ASCII log format)
$RTCM	Injects RTCM format differential corrections in ASCII (Type 1)	(follows NovAtel ASCII log format)
$TM1A	Injects receiver time of 1 PPS	(follows NovAtel ASCII log format)
$UTCA	Injects UTC information	(follows NovAtel ASCII log format)
ACCEPT	Port input control (set command interpreter)	**accept** *port,option*
ANTENNAPOWER	Power to the low-noise amplifier of an active antenna	**antennapower** *flag*
ASSIGN	Assign a prn to a channel #	**assign** *channel,prn,doppler, search window*
UNASSIGN	Un-assign a channel	**unassign** *channel*
UNASSIGNALL	Un-assign all channels	**unassignall**
CLOCKADJUST	Disable clock steering mechanism	**clockadjust** *switch*
COMn	Initialize Serial Port (1 or 2)	**comn** *bps,parity,databits,stopbits, handshake,echo*
COMn_DTR	Programmable DTR lead/tail time	**comn_dtr** *control,active,lead,tail*
COMn_RTS	Programmable RTS lead/tail time	**comn_rts** *control,active,lead,tail*
CONFIG	Switches the channel configuration of the GPSCard	**config** *cfgtype*
CRESET	Configuration reset to factory default	**creset**
CSMOOTH	Sets carrier smoothing	**csmooth** *value*
DATUM	Choose a DATUM name type	**datum** *option*
USERDATUM	User defined DATUM	**userdatum** *semi-major,flattening,dx,dy,dz, rx,ry,rz, scale*
DGPSTIMEOUT	Sets maximum age of differential data to be accepted and ephemeris delay	**dgpstimeout** *value value*
DIFF_PROTOCOL	Differential correction message encoding and decoding for implementation in the GPS card firmware	**diff_protocol** *type key* or **diff_protocol** *disable* or **diff_protocol**
DYNAMICS	Set receiver dynamics	**dynamics** *option [user_dynamics]*
ECUTOFF	Set elevation cutoff angle	**ecutoff** *angle*
EXTERNALCLOCK	Sets default parameters of an optional external oscillator	**externalclock** *option*
EXTERNALCLOCK FREQUENCY	Sets clock rate	**external frequency** *clock rate*
FIX HEIGHT	Sets height for 2D navigation	**fix height** *height [auto]*
FIX POSITION	Set antenna coordinates for reference station	**fix position** *lat,lon,height [station id] [health]*
FIX VELOCITY	Accepts INS xyz (ECEF) input to aid in high velocity reacquisition of SVs	**fix velocity** *vx,vy,vz*
UNFIX	Remove all receiver FIX constraints	**unfix**
FREQUENCY_OUT	Variable frequency output (programmable)	**frequency_out** *n,k*
FRESET	Clears all data which is stored in non-volatile memory	**freset**
HELP or ?	On-line command help	**help** *option* or **?** *option*
LOCKOUT	Lock out satellite	**lockout** *prn*
UNLOCKOUT	Restore satellite	**unlockout** *prn*
UNLOCKOUTALL	Restore all satellites	**unlockoutall**
LOG	Choose data logging type	**log** *[port],datatype,[trigger],[period],[offset],[hold]*
UNLOG	Disable a data log	**unlog** *[port],data type*
UNLOGALL	Disable all data logs	**unlogall** *[port]*
MAGVAR	Set magnetic variation correction	**magvar** *value*
MESSAGES	Disable error reporting from command interpreter	**messages** *port,option*
POSAVE	Implements position averaging for reference station	**posave** *maxtime, maxhorstd, maxvertstd*
RESET	Performs a hardware reset (OEM only)	**reset**
RINEX	Configure the user defined fields in the file headers	**rinex** *cfgtype*
RTCM16T	Enter an ASCII text message to be sent out in the RTCM data stream	**rtcm16t** *ascii message*
RTCMRULE	Set variations of the RTCM bit rule	**rtcmrule** *rule*
RTKMODE	Set up the RTK mode	**rtkmode** *argument, data range*

SAVEALMA	Save the latest almanac in non-volatile memory	**savealma** *option*
SAVECONFIG	Save current configuration in non-volatile memory (OEM only)	**saveconfig**
SEND	Send an ASCII message to any of the communications ports	**send** *port ascii-message*
SENDHEX	Sends non-printable characters in hexadecimal pairs	**sendhex** *port data*
SETDGPSID	Enter in a reference station ID	**setdgpsid** *option*
SETHEALTH	Override PRN health	**sethealth** *prn,health*
RESETHEALTH	Reset PRN health	**resethealth** *prn*
RESETHEALTHALL	Reset all PRN health	**resethealthall**
SETL1OFFSET	Add an offset to the L1 pseudorange to compensate for signal delays	**setL1offset** *distance*
SETNAV	Set a destination waypoint	**setnav** *from lat,from lon,to lat, to lon,track offset, from port,to port*
SETTIMESYNC	Enable or disable time synchronization	**settimesync** *flag*
UNDULATION	Choose undulation	**undulation** *separation*
VERSION	Current software and hardware information	**version**

Appendix B – Developed Software

```
File name:  GPSApplet[version].java

Purpose:    To provide a friendly graphical users for users to enter
            Command and to track the Rover receiver position.


//<applet code=GPSApplet.class height=800 width=400>
//<By Andy Au and Donald Ying>

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.*;

public class GPSApplet30 extends Applet implements ActionListener {

  final int port = 6789;  //make this same as server port
  String servername = "localhost";
  Socket clientSocket;
  DataOutputStream toServer;

  //Grafix stuff
  Label latitudeData, longitudeData, heightData, latitudeSDData,
longitudeSDData, heightSDData;
  Button bButtonOne, bButtonTwo, bButtonThree, bButtonFour, bButtonFive,
bButtonSix, b;
  TextField inputText;
  Label label, label2, labelOne, labelTwo, labelThree, labelFour,
labelFive, labelSix, labelSeven;
  int fileReadLine;

  //constants
  static final String connect = " Connect!!  ";

  public void init() {

    bButtonOne   = new Button("Fix Master Receiver location");
    bButtonTwo   = new Button("Logging Master Receiver data");
    bButtonThree = new Button("Set DGPS timeout");
    bButtonFour =  new Button("Receive Data from Master");
    bButtonFive =  new Button("Logging Rover receiver data");
    bButtonSix  =  new Button("Display Rover Location");
    b = new Button("Submit Command");

    label      = new Label();
    labelOne   = new Label();
    labelTwo   = new Label();
    labelThree = new Label();
    labelFour  = new Label();
    labelFive  = new Label();
    labelSix   = new Label();
    labelSeven = new Label();
```

```
latitudeData    = new Label("Latitude:");
longitudeData   = new Label("Longitude:");
heightData      = new Label("Height:");
latitudeSDData  = new Label("Standard Deviation:");
longitudeSDData = new Label("Standard Deviation:");
heightSDData    = new Label("Standard Deviation:");
labelSeven      = new Label("e.g. unlogall, unfix, etc.");

String message = "";
inputText = new TextField(message, 15 );
fileReadLine = 0;

setLayout(new GridLayout(17, 3));
add(new Label());
add(new Label("     [ GPSCard APPLET CONTROLLER ]   "));
add(new Label());
add(new Label("==== Control master receiver ===="));
add(new Label());
add(new Label());
add(bButtonOne);
add(bButtonTwo);
add(bButtonThree);
add(labelOne);
add(labelTwo);
add(labelThree);
add(new Label());
add(new Label());
add(new Label());
add(new Label("====  Control rover receiver ===="));
add(new Label());
add(new Label());
add(bButtonFour);
add(bButtonFive);
add(bButtonSix);
add(labelFour);
add(labelFive);
add(labelSix);
add(new Label());
add(new Label());
add(new Label());
add(new Label("=== Manually enter command here ==="));
add(new Label());
add(new Label());
add(inputText);
add(b);
add(new Label());
add(labelSeven);
add(new Label());
add(new Label());
add(new Label());
add(new Label());
add(new Label());
add(new Label("=== RTK GPS Rover reciever location ==="));
add(new Label());
add(new Label());
add(latitudeData);
add(longitudeData);
add(heightData);
add(latitudeSDData);
add(longitudeSDData);
```

```
    add(heightSDData);
    add(new Label());
    add(new Label());
    add(new Label());

    bButtonOne.addActionListener(this);
    bButtonTwo.addActionListener(this);
    bButtonThree.addActionListener(this);
    bButtonFour.addActionListener(this);
    bButtonFive.addActionListener(this);
    bButtonSix.addActionListener(this);
    b.addActionListener(this);
  }


  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == bButtonOne) {


      SimpleWrite sw = new SimpleWrite();
      sw.messageString = "fix position -33.91776927 -115.2318233055
86.983";
      labelOne.setText("Submitted: fix position -33.9 -115.2 86.9");
      labelTwo.setText(" ");
      labelThree.setText(" ");
      labelFour.setText(" ");
      labelFive.setText(" ");
      labelSix.setText(" ");
      labelSeven.setText("e.g. unlogall, unfix, etc.");
      latitudeData.setText("Latitude: ");
      longitudeData.setText("Longitude: ");
      heightData.setText("Height: ");
      latitudeSDData.setText("Standard Deviation: ");
      longitudeSDData.setText("Standard Deviation: ");
      heightSDData.setText("Standard Deviation: ");
    }
    else if (e.getSource() == bButtonTwo) {

      SimpleWrite sw = new SimpleWrite();
      sw.messageString = "log com2 RTCAREF ontime 10";
      labelOne.setText(" ");
      labelTwo.setText("Submitted: log com2 RTCAREF ontime 10");
      labelThree.setText(" ");
      labelFour.setText(" ");
      labelFive.setText(" ");
      labelSix.setText(" ");
      labelSeven.setText("e.g. unlogall, unfix, etc.");
      latitudeData.setText("Latitude: ");
      longitudeData.setText("Longitude: ");
      heightData.setText("Height: ");
      latitudeSDData.setText("Standard Deviation: ");
      longitudeSDData.setText("Standard Deviation: ");
      heightSDData.setText("Standard Deviation: ");
    }
    else if (e.getSource() == bButtonThree) {

      SimpleWrite sw = new SimpleWrite();
      sw.messageString = "dgpstimeout 2, 300";
      labelOne.setText(" ");
      labelTwo.setText(" ");
```

```
    labelThree.setText("Submitted: dgpstimeout 2, 300");
    labelFour.setText(" ");
    labelFive.setText(" ");
    labelSix.setText(" ");
    labelSeven.setText("e.g. unlogall, unfix, etc.");
    latitudeData.setText("Latitude: ");
    longitudeData.setText("Longitude: ");
    heightData.setText("Height: ");
    latitudeSDData.setText("Standard Deviation: ");
    longitudeSDData.setText("Standard Deviation: ");
    heightSDData.setText("Standard Deviation: ");
  }

  else if (e.getSource() == bButtonFour) {

    SimpleWrite sw = new SimpleWrite();
    sw.messageString = "accept com2 rtcm";
    labelOne.setText(" ");
    labelTwo.setText(" ");
    labelThree.setText(" ");
    labelFour.setText("Submit: accept com2 rtcm");
    labelFive.setText(" ");
    labelSix.setText(" ");
    labelSeven.setText("e.g. unlogall, unfix, etc.");
    latitudeData.setText("Latitude: ");
    longitudeData.setText("Longitude: ");
    heightData.setText("Height: ");
    latitudeSDData.setText("Standard Deviation: ");
    longitudeSDData.setText("Standard Deviation: ");
    heightSDData.setText("Standard Deviation: ");
  }
  else if (e.getSource() == bButtonFive) {

    SimpleWrite sw = new SimpleWrite();
    sw.messageString = "log com1 prtka ontime 1";
    labelOne.setText(" ");
    labelTwo.setText(" ");
    labelThree.setText(" ");
    labelFour.setText(" ");
    labelFive.setText("Submitted: log com1 prtka ontime 1");
    labelSix.setText(" ");
    labelSeven.setText("e.g. unlogall, unfix, etc.");
    latitudeData.setText("Latitude: ");
    longitudeData.setText("Longitude: ");
    heightData.setText("Height: ");
    latitudeSDData.setText("Standard Deviation: ");
    longitudeSDData.setText("Standard Deviation: ");
    heightSDData.setText("Standard Deviation: ");
  }
  else if (e.getSource() == bButtonSix) {

    try {

      char[] latitude    = new char[15];
      char[] longitude   = new char[15];
      char[] height      = new char[7];
      char[] latitudeSD  = new char[6];
      char[] longitudeSD = new char[6];
      char[] heightSD    = new char[6];
      String latitudeValue    = " ";
```

```
        String longitudeValue   = " ";
        String heightValue      = " ";
        String latitudeSDValue  = " ";
        String longitudeSDValue = " ";
        String heightSDValue    = " ";

        fileReadLine = fileReadLine+1;

//        SimpleRead code = new SimpleRead();

        String[] data = new String[125];

        BufferedReader stdin = new BufferedReader(new
FileReader("GPSCard.log"));

//        BufferedReader stdin = new BufferedReader(new
InputStreamReader(code.inputStream));

        for (int i = 0; i < 125; i++) {
          String currentLine = stdin.readLine();
          if (currentLine == null) {
            break;
          }
          StringTokenizer st = new StringTokenizer(currentLine);
          data[i] = st.nextToken();
        }

        if (data[fileReadLine].charAt(1)=='P') {
          for (int i = 0; i < 15; i++) {
            latitude[i] = data[fileReadLine].charAt(33+i);
          }
          for (int i = 0; i < 15; i++) {
            longitude[i] = data[fileReadLine].charAt(49+i);
          }
          for (int i = 0; i < 7; i++) {
            height[i] = data[fileReadLine].charAt(65+i);
          }
          for (int i = 0; i < 6; i++) {
            latitudeSD[i] = data[fileReadLine].charAt(83+i);
          }
          for (int i = 0; i < 6; i++) {
            longitudeSD[i] = data[fileReadLine].charAt(90+i);
          }
          for (int i = 0; i < 6; i++) {
            heightSD[i] = data[fileReadLine].charAt(97+i);
          }
        }

        if (data[fileReadLine].charAt(1)=='R') {
          for (int i = 0; i < 15; i++) {
            latitude[i] = data[fileReadLine].charAt(26+i);
          }
          for (int i = 0; i < 15; i++) {
            longitude[i] = data[fileReadLine].charAt(42+i);
          }
          for (int i = 0; i < 7; i++) {
            height[i] = data[fileReadLine].charAt(58+i);
          }
          for (int i = 0; i < 6; i++) {
            latitudeSD[i] = data[fileReadLine].charAt(76+i);
```

```
        }
        for (int i = 0; i < 6; i++) {
          longitudeSD[i] = data[fileReadLine].charAt(83+i);
        }
        for (int i = 0; i < 6; i++) {
          heightSD[i] = data[fileReadLine].charAt(90+i);
        }
      }

      for (int i = 0; i < 15; i++) {
        latitudeValue = latitudeValue + latitude[i];
      }
      for (int i = 0; i < 15; i++) {
        longitudeValue = longitudeValue + longitude[i];
      }
      for (int i = 0; i < 7; i++) {
        heightValue = heightValue + height[i];
      }
      for (int i = 0; i < 6; i++) {
        latitudeSDValue = latitudeSDValue + latitudeSD[i];
      }
      for (int i = 0; i < 6; i++) {
        longitudeSDValue = longitudeSDValue + longitudeSD[i];
      }
      for (int i = 0; i < 6; i++) {
        heightSDValue = heightSDValue + heightSD[i];
      }

      labelOne.setText(" ");
      labelTwo.setText(" ");
      labelThree.setText(" ");
      labelFour.setText(" ");
      labelFive.setText(" ");
      labelSix.setText("Rover Receiver location is listed below");
      labelSeven.setText("e.g. unlogall, unfix, etc.");
      latitudeData.setText("Latitude:"+ latitudeValue);
      longitudeData.setText("Longitude: "+ longitudeValue);
      heightData.setText("Height: "+ heightValue);
      latitudeSDData.setText("Standard Deviation: " + latitudeSDValue);
      longitudeSDData.setText("Standard Deviation: "+ longitudeSDValue);
      heightSDData.setText("Standard Deviation: "+ heightSDValue);


    }
    catch (IOException h) {

      // System.out.println("Error: " + e.toString());

    }

  }
  else if (e.getSource() == b) {

    String command = inputText.getText();

    labelOne.setText(" ");
    labelTwo.setText(" ");
    labelThree.setText(" ");
    labelFour.setText(" ");
    labelFive.setText(" ");
```

```
      labelSix.setText(" ");

      if (command.length() == 0) {
        labelSeven.setText("ERROR: Empty Command Not Submitted");
      } else {
        labelSeven.setText("Submitted: " + command);
        SimpleWrite sw = new SimpleWrite();
        sw.messageString = command;
        inputText.setText("");

      }
      latitudeData.setText("Latitude: ");
      longitudeData.setText("Longitude: ");
      heightData.setText("Height: ");
      latitudeSDData.setText("Standard Deviation: ");
      longitudeSDData.setText("Standard Deviation: ");
      heightSDData.setText("Standard Deviation: ");
    }
  }
}
```

```
File name:  SimpleRead.java

Purpose:    To read data from the Serial Port

import java.io.*;
import java.util.*;
import javax.comm.*;

public class SimpleRead implements Runnable, SerialPortEventListener {
    static CommPortIdentifier portId;
    static Enumeration portList;

    InputStream inputStream;
    SerialPort serialPort;
    Thread readThread;

    public static void main(String[] args) {
        portList = CommPortIdentifier.getPortIdentifiers();

        while (portList.hasMoreElements()) {
            portId = (CommPortIdentifier) portList.nextElement();
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                if (portId.getName().equals("COM1")) {
                //  if (portId.getName().equals("/dev/term/a")) {
                    SimpleRead reader = new SimpleRead();
                }
            }
        }
    }

    public SimpleRead() {
        try {

        // public synchronized CommPort open(String appname, int timeout)
throws PortInUseException

        // Opens the communications port. open obtains exclusive
ownership of the port. If the port
        // is owned by some other application, a PORT_OWNERSHIP_REQUESTED
event is propagated using
        // the CommPortOwnershipListener event mechanism. If the
application that owns the port calls
        // close during the event processing, then this open will succeed.
There is one InputStream
        // and one OutputStream associated with each port. After a port
is opened with open, then all
        // calls to getInputStream will return the same stream object
until close is called.

            serialPort = (SerialPort) portId.open("GPSApplet", 2000);

        } catch (PortInUseException e) {}
        try {
            inputStream = serialPort.getInputStream();
        } catch (IOException e) {}
        try {
            serialPort.addEventListener(this);
        } catch (TooManyListenersException e) {}
        serialPort.notifyOnDataAvailable(true);
        try {
```

```
            serialPort.setSerialPortParams(9600,
                SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1,
                SerialPort.PARITY_NONE);
        } catch (UnsupportedCommOperationException e) {}
        readThread = new Thread(this);
        readThread.start();
    }

    public void run() {
        try {
            Thread.sleep(20000);
        } catch (InterruptedException e) {}
    }

    public void serialEvent(SerialPortEvent event) {
        switch(event.getEventType()) {
        case SerialPortEvent.BI:
        case SerialPortEvent.OE:
        case SerialPortEvent.FE:
        case SerialPortEvent.PE:
        case SerialPortEvent.CD:
        case SerialPortEvent.CTS:
        case SerialPortEvent.DSR:
        case SerialPortEvent.RI:
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            break;
        case SerialPortEvent.DATA_AVAILABLE:
            byte[] readBuffer = new byte[20];

            try {
                while (inputStream.available() > 0) {
                    int numBytes = inputStream.read(readBuffer);
                }
                System.out.print(new String(readBuffer));
            } catch (IOException e) {}
            break;
        }
    }
}
```

```
File name:  SimpleWrite.java
Purpose:    To write command to the Serial Port


import java.io.*;
import java.util.*;
import javax.comm.*;

public class SimpleWrite {
    static Enumeration portList;
    static CommPortIdentifier portId;
    static String messageString = "Hello, world!\n";
    static SerialPort serialPort;
    static OutputStream outputStream;

    public static void main(String[] args) {
        portList = CommPortIdentifier.getPortIdentifiers();

        while (portList.hasMoreElements()) {
            portId = (CommPortIdentifier) portList.nextElement();
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                if (portId.getName().equals("COM1")) {
                // if (portId.getName().equals("/dev/term/a")) {
                    try {
                        serialPort = (SerialPort)
                            portId.open("GPSApplet", 2000);
                    } catch (PortInUseException e) {}
                    try {
                        outputStream = serialPort.getOutputStream();
                    } catch (IOException e) {}
                    try {
                        serialPort.setSerialPortParams(9600,
                            SerialPort.DATABITS_8,
                            SerialPort.STOPBITS_1,
                            SerialPort.PARITY_NONE);
                    } catch (UnsupportedCommOperationException e) {}
                    try {
                        outputStream.write(messageString.getBytes());
                    } catch (IOException e) {}
                }
            }
        }
    }
}
```

```
File name:  GPSApplet.html
Purpose:    To display the file GPSApplet.class in a website.

<html>
<head>
<title>
  GPS Applet Control Panel
</title>
<body background="background.gif">
  <center>

    <table>
      <tr><td>
      <hr width=700><font size=+2><b><center>
        GPS Applet Control Panel
      </center></font><hr width=700>

      <applet code="GPSApplet30.class" width=700 height=500></applet>
      <p>
      </td></tr>
      </table>
  </center>
</body>
</html>
```

Appendix C – Logs Summary

```
File name:  GPSCard.log
Purpose:    Contain all the information provided by the GPS receiver.


=== Extracted from the GPSCard.log ===


$PRTKA,154,266190.00,1.000,5,5,5,-
33.91776926442,151.23182328433,86.9908,0.0000,61,0.0128,0.0157,0.0369,0,0
,4,52,1334321*4D


PRTKA Structure:
[$PRTKA] [week] [sec] [lag] [#sv] [#high] [L1L2 #high] [lat] [lon] [hgt]
[undulation] [datum ID] [lat SdDev] [lon SdDev] [hgt SdDev] [soln status]
[rtk status] [posn type] [idle] [stn ID] [*xx] [ [CR][LF] ]
```

Field #	Field type	Data Description	Example
1	$PRTKA	Log header	$PRTKA
2	week	GPS week number	872
3	sec	GPS time into the week (in seconds)	174963.00
4	lag	Differential lag in seconds	1.000
5	#sv	Number of matched satellites; may differ from the number in view.	8
6	#high	Number of matched satellites above RTK mask angle; observations from satellites below mask are heavily de-weighted	7
7	L1L2 #high	Number of matched satellites above RTK mask angle with both L1 and L2 available	7
8	lat	Latitude of position in current datum, in decimal fraction format. A negative sign implies South latitude	51.11358042429
9	lon	Longitude of position in current datum, in decimal fraction format. A negative sign implies West longitude	-114.04358006710
10	hgt	Height of position in current datum, in meters above mean sea level	1059.4105
11	undulation	Geoidal separation, in meters, where(+ve) is above ellipsoid and (-ve) is below ellipsoid	-16.2617
12	datum ID	Current datum (see *Appendix G, Page 234*)	61
13	lat σ	Standard deviation of latitude solution element, in meters	0.0096
14	lon σ	Standard deviation of longitude solution element, in meters	0.0100
15	hgt σ	Standard deviation of height solution element, in meters	0.0112
16	soln status	Solution status *(see Table D-1, Page 142)*	0
17	rtk status	RTK status (see *Tables D-3, D-4, Page 143)*	0
18	posn type	Position type *(see Table D-2, Page 143)*	4
19	idle	Percent idle time, percentage	42
20	stn ID	Reference station identification (RTCM: 0 - 1023, or RTCA: 266305 - 15179385)	119
21	*xx	Checksum	*51
22	[CR][LF]	Sentence terminator	[CR][LF]

```
File name:  GPSCard.log
Purpose:    Contain all the information provided by the GPS receiver.


=== Extracted from the GPSCard.log ===


$RTKA,154,266189.00,5,5,5,-
33.91776928983,151.23182335809,86.9710,0.0000,61,0.0074,0.0091,0.0256,0,0
,4,1,1334321*2E


RTKA Structure:
[$RTKA] [week] [seconds] [#sv] [#high] [L1L2 #high] [lat] [lon] [hgt]
[undulation] [datum ID] [lat SdDev] [lon SdDev] [hgt SdDev] [soln status]
[rtk status] [posn type] [dyn mode] [stn ID] [*xx] [ [CR][LF] ]
```

Field #	Field type	Data Description	Example
1	$RTKA	Log header	$RTKA
2	week	GPS week number	872
3	seconds	GPS time into the week (in seconds)	174962.00
4	#sv	Number of matched satellites; may differ from the number in view.	8
5	#high	Number of matched satellites above RTK mask angle; observations from satellites below mask are heavily de-weighted	7
6	L1L2 #high	Number of matched satellites above RTK mask angle with both L1 and L2 available	7
7	lat	Latitude of position in current datum, in decimal fraction format. A negative sign implies South latitude	51.11358039754
8	lon	Longitude of position in current datum, in decimal fraction format. A negative sign implies West longitude	-114.04358003164
9	hgt	Height of position in current datum, in meters above mean sea level	1059.4105
10	undulation	Geoidal separation, in meters, where positive is above ellipsoid and negative is below ellipsoid	-16.2617
11	datum ID	Current datum (see *Appendix G, Page 234*)	61
12	lat σ	Standard deviation of latitude solution element, in meters	0.0036
13	lon σ	Standard deviation of longitude solution element, in meters	0.0039
14	hgt σ	Standard deviation of height solution element, in meters	0.0066
15	soln status	Solution status (see *Table D-1, Page 142*)	0
16	rtk status	RTK status (see *Tables D-3, D-4, Page 143*)	0
17	posn type	Position type (see *Table D-2, Page 143*)	4
18	dyn mode	Dynamics mode (0= static, 1= kinematic)	0
19	stn ID	Reference station identification (RTCM: 0 - 1023, or RTCA: 266305 - 15179385)	119
20	*xx	Checksum	*33
21	[CR][LF]	Sentence terminator	[CR][LF]

Appendix D – Selected Experimental Results

University Local Area Network
UDP and TCP/IP Test Results
From: UNSW, Residential college 129.94.199.101 (100Mbps)
To: UNSW, EE Blue Tooth lab 149.171.37.185 (100Mbps)
Date: Monday 16th, Tuesday 17th and Saturday 21st September 2002

```
Data UDP 1460 Bytes Broadcast
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:24
========================================
TCP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Throughput in Mbits/s
Tx -   Min: 3.00  Max: 5.82  Avg: 5.39
Rx -   Min: 5.87  Max: 6.82  Avg: 6.55
========================================
UDP Packet Error Rate %
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 11.00  Avg: 2.11
========================================
```

```
Data TCP 1.0 MByte
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:39
========================================
TCP Throughput in Mbits/s
Tx -   Min: 8.58  Max: 22.71  Avg: 18.52
Rx -   Min: 0.65  Max: 52.77  Avg: 46.20
========================================
UDP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Packet Error Rate %
Tx -   Min: 0.0  Max: 0.0  Avg: 0.0
Rx -   Min: 0.0  Max: 0.0  Avg: 0.0
========================================
```

```
Data UDP 1460 Bytes Unicast
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:24
========================================
TCP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Throughput in Mbits/s
Tx -   Min: 1.30  Max: 6.87  Avg: 5.83
Rx -   Min: 6.84  Max: 7.45  Avg: 7.25
========================================
UDP Packet Error Rate %
Tx -   Min: 0.05  Max: 32.1  Avg: 6.10
Rx -   Min: 0.00  Max: 4.00  Avg: 0.40
========================================
```

```
Data TCP 2.0 MByte
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:29
========================================
TCP Throughput in Mbits/s
Tx -   Min: 11.81  Max: 23.97  Avg: 21.56
Rx -   Min: 46.96  Max: 55.42  Avg: 53.84
========================================
UDP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Packet Error Rate %
Tx -   Min: 0.0  Max: 0.0  Avg: 0.0
Rx -   Min: 0.0  Max: 0.0  Avg: 0.0
========================================
```

```
Data UDP 512 Bytes Broadcast
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:22
========================================
TCP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Throughput in Mbits/s
Tx -   Min: 0.73  Max: 3.39  Avg: 3.01
Rx -   Min: 3.78  Max: 4.06  Avg: 3.98
========================================
UDP Packet Error Rate %
Tx -   Min: 0.00  Max: 19.00  Avg: 1.90
Rx -   Min: 0.00  Max: 3.00  Avg: 0.30
========================================
```

```
Data TCP 3.0 MByte
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:34
========================================
TCP Throughput in Mbits/s
Tx -   Min: 13.59  Max: 24.56  Avg: 22.56
Rx -   Min: 48.11  Max: 55.84  Avg: 54.32
========================================
UDP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Packet Error Rate %
Tx -   Min: 0.0  Max: 0.0  Avg: 0.0
Rx -   Min: 0.0  Max: 0.0  Avg: 0.0
========================================
```

```
Data UDP 512 Bytes Unicast
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:22
========================================
TCP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Throughput in Mbits/s
Tx -   Min: 1.62  Max: 4.31  Avg: 3.37
Rx -   Min: 4.31  Max: 4.73  Avg: 4.53
========================================
UDP Packet Error Rate %
Tx -   Min: 0.00  Max: 8.70  Avg: 2.30
Rx -   Min: 0.00  Max: 2.00  Avg: 0.20
========================================
```

```
Data UDP 256 Bytes Broadcast
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:22
========================================
TCP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Throughput in Mbits/s
Tx -   Min: 0.83  Max: 2.27  Avg: 2.02
Rx -   Min: 2.73  Max: 2.94  Avg: 2.84
========================================
UDP Packet Error Rate %
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
```

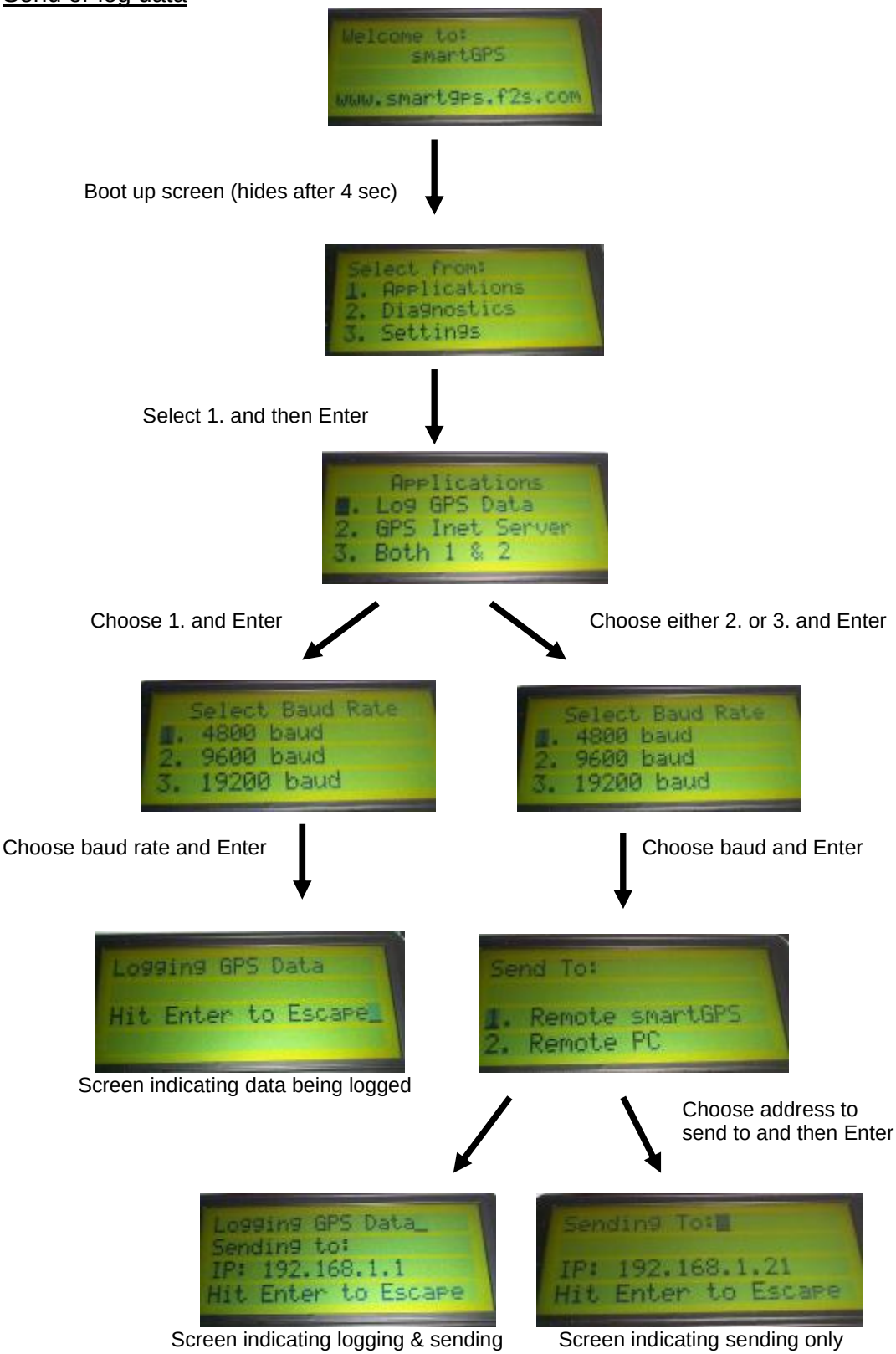
```
Data UDP 256 Bytes Unicast
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:22
========================================
TCP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Throughput in Mbits/s
Tx -   Min: 2.66  Max: 2.81  Avg: 2.84
Rx -   Min: 2.94  Max: 3.54  Avg: 3.39
========================================
UDP Packet Error Rate %
Tx -   Min: 0.00  Max: 1.51  Avg: 0.60
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
```

```
Data TCP 4.0 MByte
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:40
========================================
TCP Throughput in Mbits/s
Tx -   Min: 13.92  Max: 24.04  Avg: 21.69
Rx -   Min: 51.34  Max: 54.83  Avg: 53.39
========================================
UDP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Packet Error Rate %
Tx -   Min: 0.0  Max: 0.0  Avg: 0.0
Rx -   Min: 0.0  Max: 0.0  Avg: 0.0
========================================
```

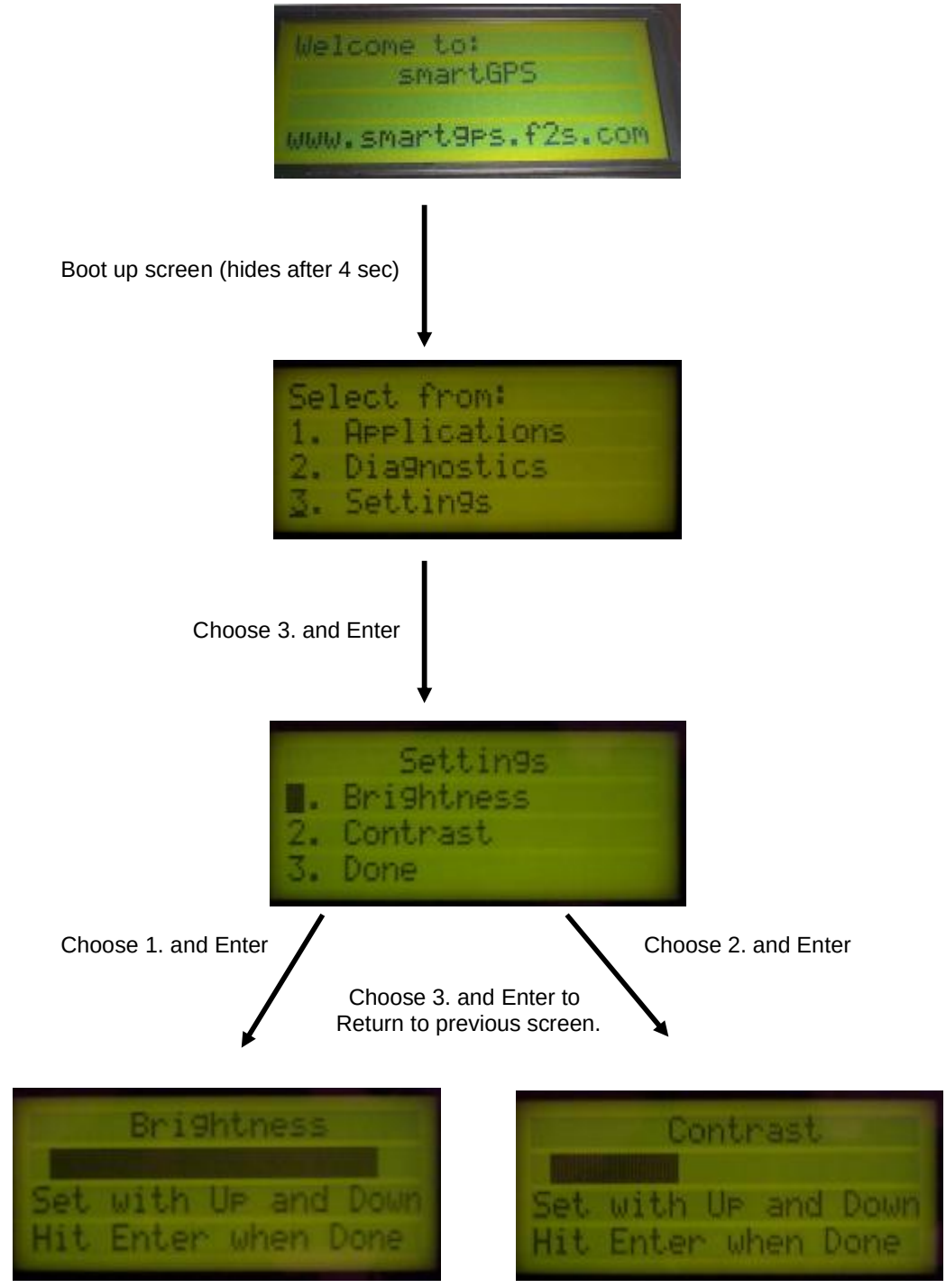
```
Data TCP 5.0 MByte
                All Test Summary
========================================
Tests Completed: 10
Elapsed Time: 00:00:48
========================================
TCP Throughput in Mbits/s
Tx -   Min: 7.97  Max: 23.37  Avg: 21.38
Rx -   Min: 53.44  Max: 55.16  Avg: 54.24
========================================
UDP Throughput in Mbits/s
Tx -   Min: 0.00  Max: 0.00  Avg: 0.00
Rx -   Min: 0.00  Max: 0.00  Avg: 0.00
========================================
UDP Packet Error Rate %
Tx -   Min: 0.0  Max: 0.0  Avg: 0.0
Rx -   Min: 0.0  Max: 0.0  Avg: 0.0
========================================
```

Appendix E – Operate smartGPS TINI board

Send or log data



Control the brightness of the LCD screen



General Setting of the smartGPS TINI